LA-UR-99-1980

Title: **Model Inversion Using Bayesian Inference And Genetic Algorithms**

Author(s): Brian J. Reardon

Submitted to:

http://lib-www.lanl.gov/la-pubs/00326786.pdf

Los Alamos
NATIONAL LABORATORY

Form No. 836 R5
ST 2629 10/91

Model Inversion Using Bayesian Inference And Genetic Algorithms

Brian J. Reardon, MST-6, Los Alamos National Laboratory, Los Alamos, NM 87545

Abstract

The intimate relationship between Bayesian statistics and genetic algorithms (GA) is elucidated and it is shown that the GA uses Bayes' rule to select members for the crossover operator and thus a GA can be formally described as a Bayesian Inference Engine (BIE). Additionally, the previously used fuzzy logic based multiple objective selection procedure has been modified to make it simpler to implement and congruent with a formal execution of Bayes' rule. The subroutines necessary for this selection procedure are provided in Appendix A. Finally, the framework of Bayesian inference has been used to better extract information out of the evolved population of a GA. Since the output of the GA is the posterior probability density (PPD) of the optimization problem, the *a posteriori* covariance matrix, C_m, can be derived and the eigenvalues ov C_m provide an important measure of performance of the GA. Likewise, the PPD itself efficiently suppresses much of the noise generated in the population due to the GA's inherently stochastic nature. The code for the BIE is provided in Appendix B.

1.0 Introduction

1.1 Inverse and Ill Posed Problems in Materials Science and Engineering

There is an ever increasing need in materials science and engineering to fit the parameters of models, which are to be used in a predictive capacity, using underdetermined experimental data sets. Model inversion of this type falls under the general category of inverse and ill – posed problems and can often be cast into the framework of Bayesian statistics (Tarantola, 1987). Such problems include determining powder densification models from limited density data, chemical potential determination from limited phase diagram data containing a high degree of uncertainty, and mechanical threshold strength (MTS) determination from mechanical tests also with a high degree of uncertainty. In all of these examples, model parameters must be optimized using limited and uncertain data sets that leave the inversion underdetermined. Likewise, if the models are to be used in a predictive capacity, there is a need to be able to quantify the expected deviation of the model from reality.

This report shows how a fuzzy logic based multi-objective genetic algorithm (GA) (Reardon 1997a-b, 1998a-e) can be used as a Bayesian Inference Engine (BIE) to evolve a posterior probability density (PPD) of the model parameter vector space:

$$M_i = \{m_1, m_2, m_3, \ldots, m_N\}^T \qquad \text{Eq. 1}$$

where M_I is a particular model to be tested, m_J is one of the N parameters used in the model and T signifies the transpose of the vector. The GA evolves a set or population of M_I's which effectively defines the PPD. Once the PPD has been sufficiently determined by the GA, parameter vectors are selected and used in the physics of the forward problem, for future experimental conditions, to evaluate the predictive capacity of the model.

As a test of the GA's ability as a BIE and for the sake simplicity, Schaffer's F2 problem will be addressed in this report. Schaffer's F2 problem is a classical

multiobjective optimization problem where the objective is to identify the range of x which sufficiently minimizes both $F_{11}(x)=x^2$ and $F_{22}(x)=(x-2)^2$. In this problem the parameter vector consists of

$$M_i = \{x_1, x_2\}^T \qquad \text{Eq. 2}$$

where $-6.0 \le x_1 \le 6.0$ is used to evaluate $F_{11}(x)$ and $F_{22}(x)$ and $0.0 \le x_2 \le 1.0$ is a dummy variable not used in any function but rather assists in evaluation of the performance of the GA. The main evaluation carried out by x_2 is for evidence of genetic drift.

1.2 Bayesian Statistics in Model Inversion

Consider a model parameter vector such as the one defined in Eq. 2 and also consider a data vector defined as:

$$D=\{d_1, d_2, d_3,\ldots,d_{ND}\}=\{0\pm4, 0\pm4\}^T \qquad \text{Eq. 3}$$

where ND is the total number of experimentally derived data points. In this example there are two data points whose value is 0±4. The goal of Bayesian analysis is to come up with a way of accepting or rejecting a particular model (M) or hypothesis given an experimental data set (D) and prior knowledge about the problem. Thus, in Bayesian statistics, the model or hypothesis is assigned a probability of acceptance and the total probability distribution function (PDF) of a series of models being tested makes up what is commonly called the posterior probability density (PPD). This goal is achievable through the central tenant of Bayesian statistics: Bayes' Rule:

$$\sigma(M \mid D) = \frac{P(M,D)}{P(D)} \qquad \text{Eq. 4}$$

which is essentially the definition of conditional probability. This rule was first proposed by Rev. Thomas Bayes and published posthumously in 1763 but has been ignored up

until the last 20 years due to the computational difficulties in evaluating the probability integrals (Bayes, 1763). This theorem says that the conditional probability of a model being correct given a set of data is a ratio of the pdf of M and D to the pdf of D alone. The numerator can be expressed as:

$$P(M,D) = P(D \mid M)P(M) \qquad \text{Eq. 5}$$

where the term P(D | M) is not a pdf but a likelihood function. Thus, while the individual components of P(D | M) are probabilities, the function itself does not integrate to 1.0. The denominator is often expressed as:

$$P(D) = \int P(D,M)dM. \qquad \text{Eq. 6}$$

Therefore, the PPD is redefined as:

$$\sigma(M \mid D) = \frac{P(D \mid M)P(M)}{\int P(D,M)dM}. \qquad \text{Eq. 7}$$

Bayes' rule as written above differs considerably from classical frequentist statistics because of the dependence of the PPD on the prior PDF, P(M). P(M) often contains subjective information about the problem that the experimentalist has *a priori*. Another major departure from frequentist statistics is the way the PPD is updated as new experimental data becomes available. The frequentist view point is that P(D) should be considered an unchanging distribution and alos that it is inappropriate to try to assign a probability of correctness to a hypothesis.

Consequently, Bayes' Rule provides the scientist with a tool that classical statistics is not capable or providing, namely, a mathematical formalization of the scientific method. When a phenomenon is observed, a hypothesis explaining the event is created often with the observer's own bias and experience in mind. This hypothesis

is then tested against new experimental data and if the data supports the hypothesis then the belief in or probability of acceptance of the hypothesis increases. An excellent introduction to the Bayesian approach to hypothesis testing can be found in Chapter 4 of Antelman (1997).

The main difficulty in using Bayes' rule, lays in the evaluation of the denominator:

$$P(D) \int P(D,M) dM, \qquad \text{Eq. 8}$$

where the integral is formally carried over the entire N-dimensional model parameter space. The accurate and fast approximation of the integration of these N-dimensional, discontinuous pdf's is the topic of many papers. Duijndam (1988a, 1988b) discussed the used of Bayes' Rule in model inversion and accomplished the above integration by assuming the PPD had a Gaussian shape then optimized the Gaussian parameters using least squares. Unfortunately, most PPD's are not Gaussian in nature and thus other techniques were needed. These techniques include Monte Carlo integration, Gibb's Sampling, and genetic algorithms (Sen and Stoffa ,1992, 1996; Sen et al., 1993; Mallick, 1995; Gerstoft, 1998).

The PPD is itself a difficult function to visualize due to its multidimensionality and its change with every new experimental data point. However, once the PPD is derived, regardless of the method, a number of important parameters describing it can be easily calculated.

The mean model can be calculated using the following formula which is a standard definition in most statistics books:

$$\langle M \rangle = \int M \sigma(M \mid D) dM \qquad \text{Eq. 9}$$

which for computational purposes is often approximated by summing over a binned PPD:

$$\langle M \rangle = \sum M\sigma(M \mid D) \qquad \text{Eq. 10}$$

Likewise, the a posteriori model covariance matrix is given by:

$$C_M = \int (M - \langle M \rangle)(M - \langle M \rangle)^T \sigma(M \mid D) dM. \qquad \text{Eq. 11}$$

The covariance matrix, which is often expressed for computational purposes as:

$$C_M = \sum MM^T \sigma(M \mid D) - \langle M \rangle \langle M \rangle^T, \qquad \text{Eq. 12}$$

provides a number of useful parameters. The standard deviation associated with the mean model is obtained through the square roots of the diagonal elements of C_M. Normalization of C_M through:

$$C_{ij} = \frac{C_{ij}}{\sqrt{C_{ii}}\sqrt{C_{jj}}} \qquad \text{Eq. 13}$$

produces the correlation matrix. A correlation coefficient of zero indicates no correlation between two variables. A positive value indicates a positive correlation and likewise a negative value indicates a negative correlation.

With C_M determined, a principle component analysis (PCA) will provide valuable insight on how well the GA is converging and what model parameters are most significant or sensitive. In PCA the data of the C_M is transformed into a new set of axes of the same number which are orthogonal to each other and are ordered based on the variance associated with that axis. The principle components of C_M can be obtain by computing its set of eigenvalues (Λ) and corresponding orthogonal eigenvectors (U) such that:

$$C_M = U\Lambda U^T \qquad \text{Eq. 14}$$

is satisfied. In a d-dimensional variable space there are d eigenvalues or principle components. However, many principle components may have small variances and thus the intrinsic dimensionality of C_M is k where k<d.

In the context of a PPD evolved by a GA, PCA is a powerful tool that assists in overcoming many deficiencies in GA's. First, as the GA evolves the population, the eigenvalues of C_M asymptotically approach specific values. When the rate of convergence reaches an acceptable minimum the GA can be stopped. Second, the largest eigenvalues and their corresponding eigenvectors indicate the most significant variables or groups of variables in the model given the available data. Thus PCA provides a sensitivity analysis of the variables in the model. Traditionally, the ability to conduct a sensitivity analysis with a GA has been hampered by the inherently stochastic nature of the GA optimization methodology. It will be shown in this paper that the formulation of the PPD and the extraction of the eigenvalues from C_M effectively filter out the stochastic noise of the GA and thus allows for an accurate determination of multi-parameter sensitivity.

Once a PPD has been determined to be reliable based on the stabilization of <M> and the quantities obtained from C_M, an optimum model can be selected from <M> assuming the correlations are properly accounted for during the selection.

1.3 Genetic Algorithms in Model Inversion and Parameter optimization

A detailed account of how a GA operates has been provided elsewhere (Reardon 1998a, b, d). In short, a GA randomly generates a set or population of parameter vectors M_i's where i = 1 ➔ N and N is the population size. This initial selection, which occurs within parameter ranges set by the user, constitutes the *a priori* information used in Bayes' Theorem. From this set, parameter vectors that satisfactory solve the optimization problem are selected. The selected members are allowed to exchange genetic material and thus produce offspring that may undergo a small degree of mutation before being placed in the next generation. Once the next generation is filled the GA starts over with selection, crossover and mutation.

The strength and novelty of the GA presented in the references (Reardon 1998a, b, d) lays in the way selection of members is conducted when dealing with multiple, conflicting, poorly defined objectives. The actual selection procedure has been modified slightly from that discussed in the references and thus for completeness the algorithm and codes will be presented here in detail.

The new selection procedure continues to use a fuzzy logic normalization scheme as well as continuously updated phenotypic niching but with a few notable changes. These changes allow for seamless connection between the GA selection operator and Bayes' Theorem.

First each parameter vector is used in the evaluation of each objective function $f_j(M_i)$ where j = 1➔N_D (N_D: number of objectives or experimental data points). The outcome of the objective function call is then compared to the experimental data using the fuzzy rule set of Figure 1 to obtain a scaled fuzzy fitness value $f_j'(f_j(M_i))$. In Figure 1, D_j is the experimentally observed data point and E_j is the uncertainty associated with D_j. $f_{j\text{-max}}$ is the maximum value for objective j in the entire population and $f_{j\text{-min}}$ is the minimum. The fuzzy fitness $f_j'(f_j(M_i))$ is obtained by finding where $f_j(M_i)$ lays on the x axis of Figure 1 and assigning its corresponding y-axis value. The total fuzzy fitness of M_I then is defined as:

$$F_T(M_i) = \frac{1}{N_D}\sum_{j=1}^{N_D} f_j'\left(f_j(M_i)\right) \qquad \text{Eq. 15}$$

where D is the number of objectives. This relation provides the fitness of a model vector as a number between 0 and 1 where 1 is the most fit.

The code that accomplishes this calculation is provided in Appendixes A.1, A.2 and A.3.

Once the scaled fitness of all the parameter vectors have been determined, selection can proceed. In the updated selection procedure a vector is randomly picked from the population and the accepted or rejected according to the likelihood provided by

Eq. 15. Thus if $F_T(M_i) = 1.0$ then the randomly picked member is accepted 100% of the time but if $F_T(M_i) = 0.25$ then the randomly picked member would be accepted only 25% of the time. Once two members have been accepted using this procedure they are compared. If one member has a better fitness than the other then the better fit is selected for crossover with another member that has been selected in the same way. If both members have the same fitness then selection goes to the member who is deemed least crowded according to a continuously updated phenotypic niche counting procedure describe in previous references (Reardon 1998a, b, d).

As eluded to previously, Eq. 15 and the niching operation taken together are a combination of the likelihood function and the *a priori* pdf used in the numerator of Bayes' Theorem. The sum over all models evaluated according to Eq. 15 becomes the denominator. Thus, the GA becomes an effective way of evolving and evaluating the PPD.

A more simplistic example of the connection between a GA and Bayes' theorem can be seen in the traditional roulette wheel selection method used in single objective GA's (Goldberg 1989) where the probability of selection is defined as:

$$\sigma(M_i) = \frac{f(M_i)}{\sum_{j=1}^{N} f(M_j)} \qquad \text{Eq. 16}$$

where $f(M_i)$ is the fitness of member M_I and the summation in the denominator is over the entire population. If the summation were over all models ever evaluated then the Eq. 16 would define the PPD. The *a priori* pdf is built into $f(M_i)$ since M_I can only occupy a parameter volume specified by the user before the GA is initiated.

The net conclusion of this analysis is that the GA acts as a BIE in that it uses Bayes' Theorem to select members in the population for crossover and thus the output of the GA is the PPD. The generation of a PPD now allows for many of the statistical tools available in Bayesian statistics to be used in the analysis of the output of the GA. Namely, from the PPD we can derive <M> and C_M. The beauty of this approach is that

the PPD can be generated at virtually no extra cost. Following the method outlined by Sen and Stoffa (1992). A 2-D array of M X B is reserved where M is the number of parameters and B is the number of values each variable can take (i.e. the number of bins). For each model at each generation an unnormalized PPD, σ(M), is computed and stored in the proper position in the bin array for each model parameter comprising each model. At the end of the GA run the model parameter PPD values are normalized. Also in a vector of length M, each component of M σ(M) is stored and summed with the correspond values from the other models. This vector provides <M>. C_M is determined by summing up MM^T σ(M) in a square array of MM for each model and at the end of the run subtracting $<M><M>^T$. The FORTRAN 90 code used to evaluate these quantities is listed in Appendix B.

Once the PPD, <M>, and C_M have been sufficiently determined, the GA can be stopped and optimal model parameter vectors can be selected and used in the physics of the forward problem for conditions that have not been experimentally tested.

2.0 Schaffer's F2 Problem

Schaffer's F2 problem, described earlier, is the classic multiple objective optimization problem where a population must evolved towards the Pareto optimal frontier. It is also a very tractable problem to visualize and thus serves as an ideal set of test functions for the correctness of an optimization theory and corresponding algorithms. For these reasons the F2 problem is explored in this work.

The optimization carried out here will involve two variables. X_1 will be used in the functions to be optimized and X_2 is a dummy variable whose evolution will be tracked to quantify the performance of the GA. Additionally, the mutation rate will be varied to determine its effects on the convergence of the solution and to test the ability of the Bayesian framework to filter out the real marginal PPD signals from the stochastic noise inherit in GA optimization.

The first step in this analysis is to determine in if the newly redefined fuzzy fitness function adequately evolves the population towards the optimal frontier. Figure 2 compares the distribution of X_1 as a function of generation obtained using the previous

selection method of Reardon (1997a) and the newly revised selection method. Both methods do an adequate job of quickly optimizing the population towards the Pareto frontier and maintaining the distribution indefinitely. The optimization of Figure 2b has a larger population size than 2a which helps to explain the slower convergence. Thus, the new objective function performs as desired and is much easier to implement into the GA.

Figures 3a-h show the scatter plots for both X_1 and X_2 as a function of generation for the four different mutation rates using the new fuzzy fitness function. The most notable artifact of figures 3a-h is that X_1 tends to converge whereas X_2 does not. This would be expected since X_1 is used in the functions and X_2 is not. At low mutation rates, diversity in the population is maintained almost exclusively by niching. Thus the distribution of the dummy variable X_2 as well as X_1 helps to assure that the selection procedure is operating in a desirable fashion. The highest mutation rate p_m=1/10 is the easiest to visually detect. Figure 3a shows significant scatter due to the high value for p_m compared to 3c, e, and g. Likewise, figure 3b seems to be much more evenly spread out compared to 3d, f, and h. There appears not to be a significant difference among the p_m's of 1/100, 1/1000, and 1/10000. However, one particularly disconcerting observation is the fact that if this were not such a simple problem and if one did not fully appreciate the impact of different mutation rates, then one may look at figure 3a and come up with an acceptable distribution for X_1 that was significantly incorrect. Even with a low p_m this same conclusion could be drawn from a visual inspection of the scatter plots or corresponding stack histograms (Figure 3i). Consequently, a more rigorous data analysis technique is needed.

The data analysis framework of Bayesian statistics which provides the marginal PPD's for each variable will be discussed here. Figure 4a-x show the PPD's for X_1 and X_2 for the 4 different p_m's at generations 0, 10 and 100. The highest p_m (1/10) PPD's are displayed in Figures a-f. From the very beginning the PPD of X_1 seems to have some structure compared to the random nature of the X_2 PPD. By generation 10 the Pareto frontier becomes quite clear but there is still a shoulder between for $2 \le X_1 \le 5$. By

generation 100 the frontier between 0 and 2 is clear but the GA still seems to identify a finite optimal probability between –6.0 and –4.0 and also between 2.0 and 5.0. However, this probability is considerably lower than what one would have gathered from figure 3a or its corresponding stack histogram. X_2 remains pretty much random throughout the optimization as would be expected.

At the lower p_m of 1/100 the PPD's are displayed in figures 4g-l. The PPD for X_1 is much crisper at generation 10 than for the high p_m study. Likewise, at generation 100 the PPD is clearly defined. X_2 remains random although there is a wider distribution in the peak heights as compared to the high p_m study.

As the p_m continues to decrease to 1/1000 as shown in figures 4m-r, X_2 again continues to remain random with a moderate increase in peak height distribution but the X_1 PPD is even more clearly defined

For a p_m of 1/10000 the PPD continues to be more crisply defined for X_1 and random for X_2. In fact it is difficult to argue any real difference between the effects of a p_m of 1/100, 1/1000, and 1/10000. With a p_m of 1/10000, hardly any genetic diversification is occurring within the population and thus one would expect a traditional GA population to experience genetic drift towards a single point within the optimal range. The reason this is not observed here is due to the niching factor present in the selection criteria.

With the PPD in hand, the mean marginal model <M> can be determined. Figures 5a-b show the mean values for X_1 and X_2 for the different mutation rates. In general, all $< X_1 >$'s approach a value of 1.0 and all $< X_2 >$'s approach a value of 0.5 which is consistent with expected behavior. The slight differences in the <M>'s are consistent with the effects of mutation observed in the scatter plots of figures 3a-i and the PPD's of figures 4a-x.

As discussed previously, the *a posteriori* covariance matrix C_M is also derived from the PPD and the square roots of the diagonal terms provide the standard deviation of the average values of figures 5a-b. Figures 6a-b show the standard deviations of figures 5a-b as a function of generation. As would be expected for the optimal range of

X_1, the standard deviations tend to converge to a value of 0.7 except for the very high p_m case. The test case p_m = 1/10 has a much higher standard deviation than the others which is to be expected from the scatter of figure 3a. The standard deviations of X_2 converge to a value close to 0.3 which is also to be expected for a random distribution between 0 and 1.

Since X_1 and X_2 are completely independent in this problem one would expect their covariance and correlation to be at or near zero. As Figures 7 and 8 show, this is in fact the case and was achieved by generation 20. In fact most of the important features of all the data previously discussed was achieve by generation 20 and then maintained through generation 100. The only obvious exception to this was the standard deviations of figure 6 which slowly converge.

The eigenvalues of C_M are a standard measure to determine how well the problem is being optimized. While many optimality conditions can be obtained from the eigenvalues and eigenvectors, the simplest would be the minimization of the sum of the eigenvalues. As shown in figure 9, the two eigenvalues of the 2 by 2 C_M in general do decrease with generation, which indicates a successful optimization.

The astute reader will notice that the eigenvalues converge towards the standard deviations of figures 6a-b. The reason for this is that the covariance of X_1 and X_2 is virtually zero and thus C_M is converging towards a diagonal matrix with each generation. Since the eigenvalues of a diagonal matrix are the square roots of the diagonal terms of the matrix this behavior is to be expected. In a C_M with nonzero off diagonal terms the eigenvalues and standard deviations would differ.

Analysis of the eigenvectors that correspond to the eigenvalues of figure 9a and 9b (see Table I) indicates that for all the mutation rates, λ_1 corresponds to the X_2 principle component and λ_2 corresponds to the X_1 principle component. Therefore, owing to the small value of λ_1 compared to λ_2 one can safely conclude that only one variable in this optimization is significant and that variable is X_1. A rather interesting item of information from Table I is that as the mutation rate (noise) increases, so does the apparent sensitivity of X_1 as indicated by λ_2. This is the result of the very high

mutation rate accentuating the lack of correlation between X_1 and X_2. This accentuation can be seen less dramatically in Figures 7 and 8.

3.0 Conclusions

This report has accomplished a number of goals. First, the intimate relationship between Bayesian statistics and genetic algorithms has been elucidated. It has been shown that the GA uses Bayes' Theorem to select members for the crossover operator and thus a GA can be formally described as a Bayesian Inference Engine. Second, the previously used fuzzy logic based multiple objective selection procedure has been modified to make it simpler to implement and congruent with a formal execution of Bayes' Theorem. The subroutines necessary for this selection procedure are provided in Appendix A. Third, the framework of Bayesian inference has been used to better extract information out of the evolved population of a GA. Since the output of the GA is the PPD of the optimization problem, the *a posteriori* covariance matrix can be derived and the eigenvalues and eigenvectors of which provide an important measure as to the performance of the GA and the sensitivity of the parameters. Likewise, the PPD itself efficiently suppresses much of the noise generated in the population due to the GA's inherently stochastic nature. This is an important accomplishment since previous sensitivity analysis attempts with GA's were unable to separate true parameter variation from the noise introduced by the stochastic nature of the GA. The code for the Bayesian inference engine is provided in Appendix B.

In more complex optimization problems, the PPD can be used to select optimal model parameter vectors. A set of these vectors can then be incorporated into the forward problem model for conditions in which experimental data is not available. The resulting average and standard deviation of the set then provides insight as to where the next experiment should be conducted.

4.0 Acknowledgments

Funded by the Department of Defense, the Department of Energy and Los Alamos National Laboratory which is operated by the University of California under contract number W-7405-ENG-36.

5.0 References

G. Antelman, 1997, “Elementary Bayesian Statistics,” Eds. A. Madansky, R. McCulloch, Edward Elgar Publishing, Inc., Lyme, NH.

T. Bayes, 1763, “An Essay towards solving a problem in the doctrine of Chances,” Philosophical Transactions of the Royal Society, 53, 370-418.

A. J. W. Duijndam, 1988a, “Bayesian Estimation in Seismic Inversion. Part I: Principles,” Geophysical Prospecting, 36, 878-898.

A. J. W. Duijndam, 1988b, “Bayesian Estimation in Seismic Inversion. Part II: Uncertainty Analysis,” Geophysical Prospecting, 36, 899-918.

P. Gerstoft and C. F. Mecklenbräuker, 1998, “Ocean Acoustic Inversion with Estimation of *a posteriori* probability distributions,” Journal of the Acoustical Society of America, 104, 2, 808-819.

D. E. Goldberg, 1989, “Genetic Algorithms in Search, Optimization, and Machine Learning,” Addison-Wesley Publishing Company, Inc., New York.

S. Mallick, 1995, “Model-based Inversion of Amplitude-variations-with-offset Data Using a Genetic Algorithm,” Geophysics, 60, 4, 939-954.

B. J. Reardon, 1998a, “Optimization of Densification Modeling Parameters of Beryllium Powder Using a Fuzzy Logic Based Multiobjective Genetic Algorithm,” Modeling and Simulation in Materials Science and Engineering, 6, 735-746, http://www.iop.org/Journals/ms

B. J. Reardon, 1998b, “Fuzzy Logic Vs. Niched Pareto Multiobjective Genetic Algorithm Optimization,” Modeling and Simulation in Materials Science and Engineering, 6, 717-734, 1998, http://www.iop.org/Journals/ms

B. J. Reardon, 1998c, “Optimization Of Densification Modeling Parameters Of Beryllium Powder Using A Fuzzy Logic Based Multiobjective Genetic Algorithm,” Los Alamos National Laboratory Unclassified Report, LA-UR-98-1036, March 1998. http://lib-www.lanl.gov/la-pubs/00412623.pdf

B. J. Reardon, 1998d, “GENES 2.0 User’s Guide: A Guide to a Fuzzy Logic Based Multiobjective Genetic Algorithm in FORTRAN 77,” Los Alamos National Laboratory Unclassified Report, LA-UR-98-3578, July 1998. http://lib-www.lanl.gov/la-pubs/00412972.pdf

B. J. Reardon, 1998e, “Optimization Of Micromechanical Densification Modeling Parameters For Copper Powder Using A Fuzzy Logic Based Multiobjective Genetic Algorithm,” Los Alamos National Laboratory Unclassified Report, LA-UR-98-0419, January 1998. http://lib-www.lanl.gov/la-pubs/00412622.pdf

B. J. Reardon, 1997a, “Fuzzy Logic Vs. Niched Pareto Multiobjective Genetic Algorithm Optimization: Part I. Schaffer’s F2 Problem,” Los Alamos National Laboratory Unclassified Report, LA-UR-97-3675, September 1997. http://lib-www.lanl.gov/la-pubs/00412620.pdf

B. J. Reardon, 1997b, “Fuzzy Logic Vs. Niched Pareto Multiobjective Genetic Algorithm Optimization: Part II. A Simplified Born-Mayer Problem,” Los Alamos National Laboratory Unclassified Report, LA-UR-97-3676, September 1997. http://lib-www.lanl.gov/la-pubs/00412621.pdf

J. D. Schaffer, 1988, “Multiple Objective Optimization with Vector Evaluated Genetic Algorithms,” Proceedings of the 1st International Conference on Genetic Algorithms and their Applications, July 24-26, Carnegie-Mellon University, Pittsburgh, Pa, Vol 1, Ed J. Gregenstette, (Hillsdale, NJ, Lawrence Erlbaum Associates) pp 93-100.

M. K. Sen, B. B. Bhattacharya, P. L. Stoffa, 1993, “Nonlinear inversion of resistively sounding data,” Geophysics, 58, 4, 496-507.

M. K. Sen and P. L. Stoffa, 1992, “Rapid sampling of model space using genetic algorithms: Examples from seismic waveform inversion,” Geophysics Journal International, 108, 281-292.

M. K. Sen and P. L. Stoffa, 1996, “Bayesian inference, Gibb’s sampler and uncertainty estimation in geophysical inversion,” Geophysical Prospecting, 44, 313-350.

A. Tarantola, 1987, “Inverse Problem Theory, Methods for Data Fitting and Parameter Estimation,” Elsevier, Amsterdam.

6.0 Tables

Table I. The final eigenvalues and corresponding eigenvectors for each mutation rate test case. Note that higher mutation rates increase the eigenvalue of λ_2 or, alternatively, make X_1 more sensitive. This is because a high mutation rate accentuates the lack of correlation between X_1 and X_2.

p_m	λ_1	ε_1	λ_2	ε_2
1/10	0.28678	2.7522E-4 -1.0000	1.5102	-1.0000 -2.7522E-4
1/100	0.28695	-3.8432E-4 -1.0000	0.72203	-1.0000 3.8432E-4
1/1000	0.27888	-5.8164E-3 -0.99998	0.72118	-0.99998 5.8164E-3
1/10000	0.29420	9.4257E-3 -0.99996	0.67292	-0.99996 -9.4257E-3

7.0 Figures

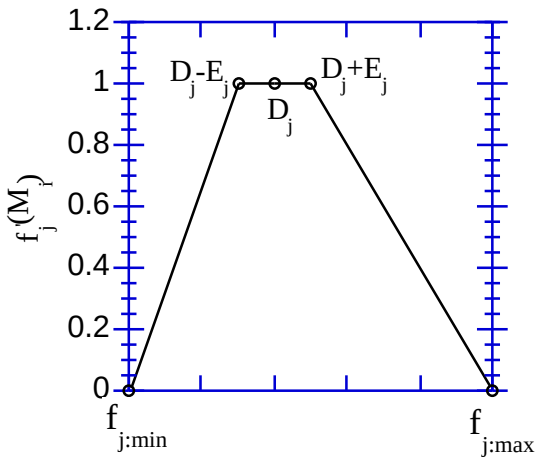


Figure 1. The fuzzy logic based fitness function.

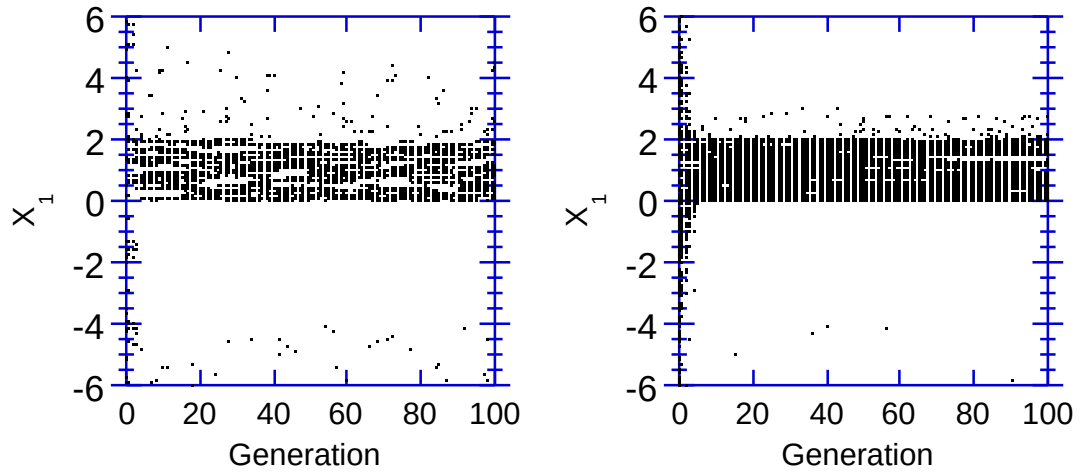
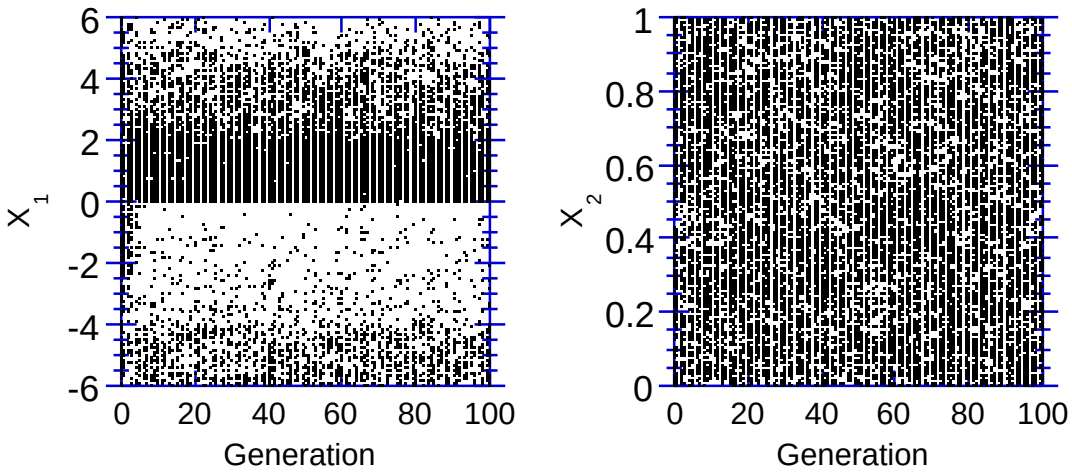
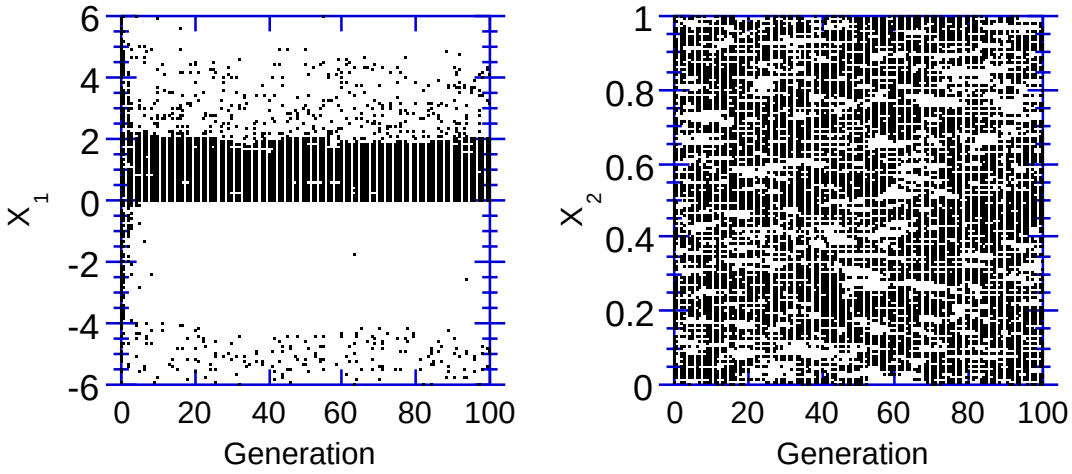


Figure 2. A scatter plot showing the distribution of X_1 as a function of generation for using the a) selection routine from Reardon (1998 a, b, d); b) the newer selection routine presented in this paper.

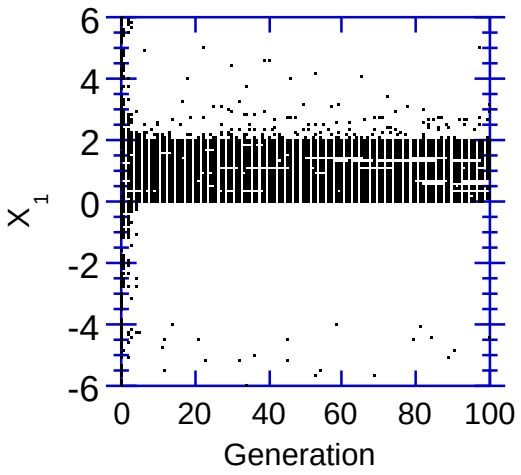


a) X_1, p_m=1 in 10; b) X_2, p_m=1 in 10;

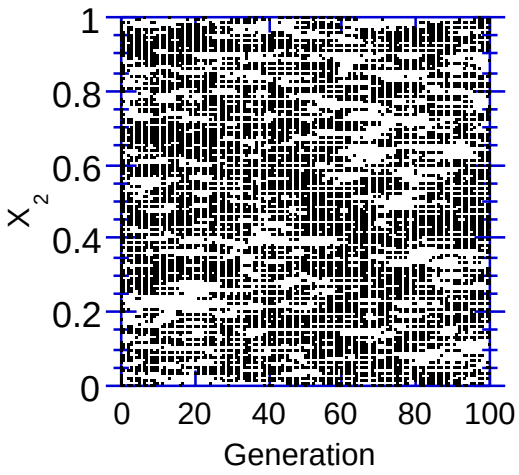


c) X_1, p_m=1 in 100; d) X_2, p_m=1 in 100;

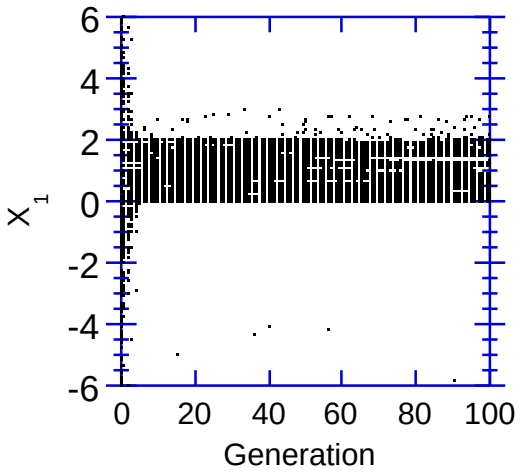
Figure 3. The scatter plots of the distribution of X_1 and X_2 as a function of generation for various mutation rates, p_m.



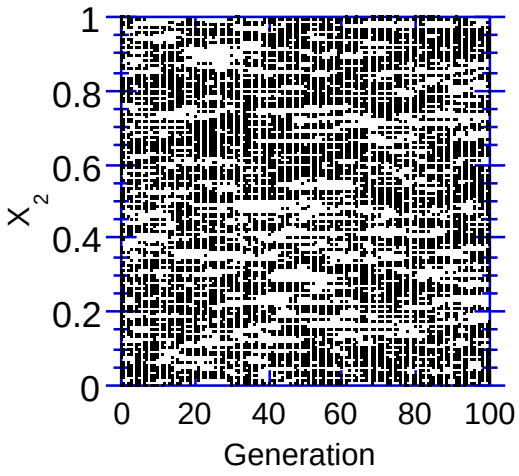
e) X_1, p_m=1 in 1000;



f) X_2, p_m=1 in 1000;

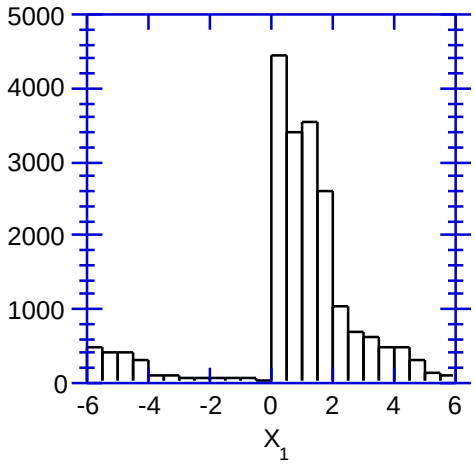


g) X_1, p_m=1 in 10000;



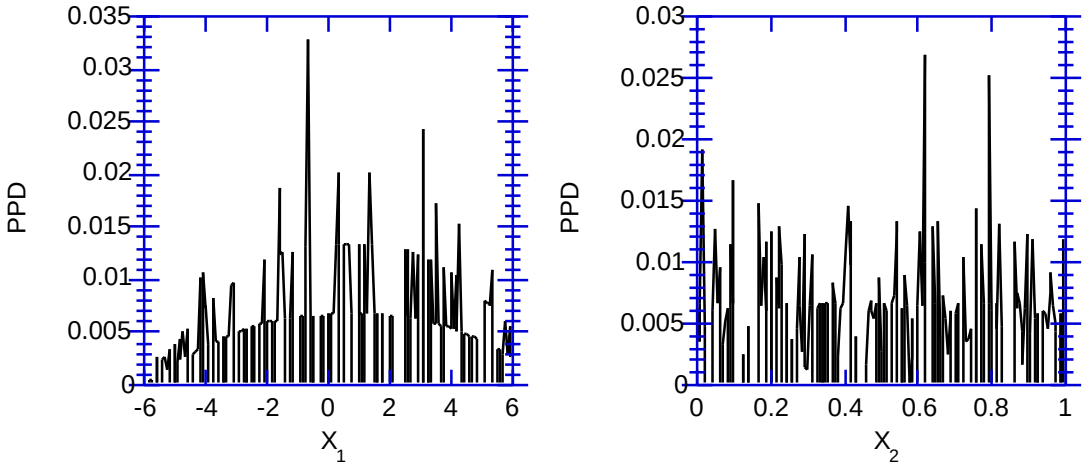
h) X_2, p_m=1 in 10000;

Figure 3. Continued.

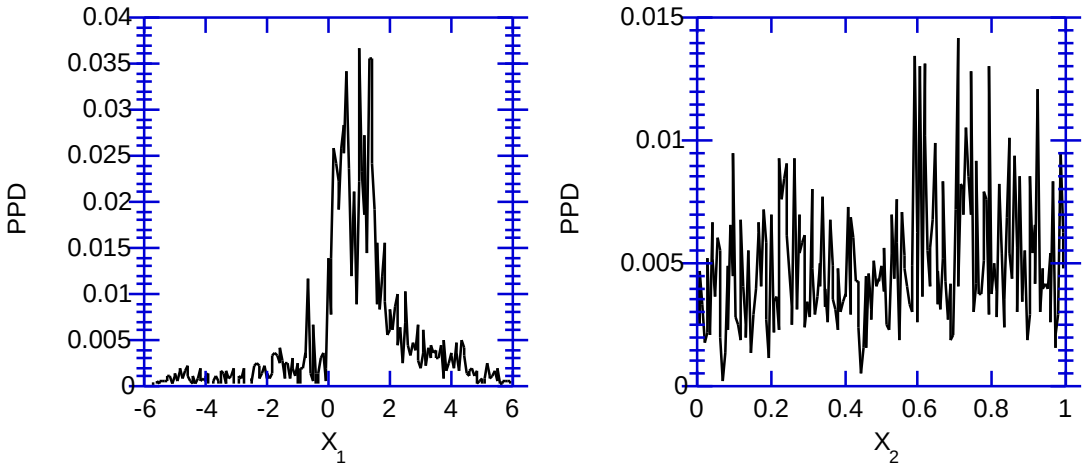


i) Stack histogram of figure 3a.

Figure 3. Continued

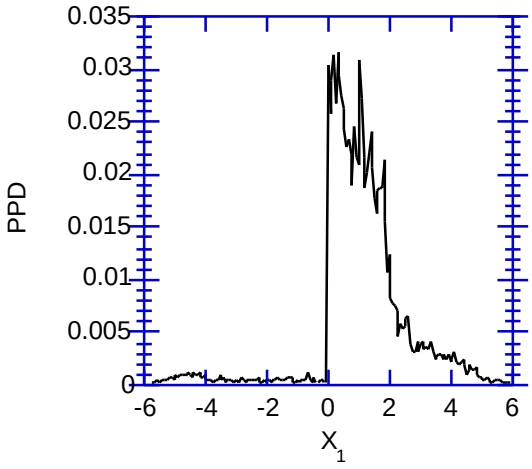


a) X_1, Gen: 0, p_m=1 in 10; b) X_2, Gen: 0, p_m=1 in 10;

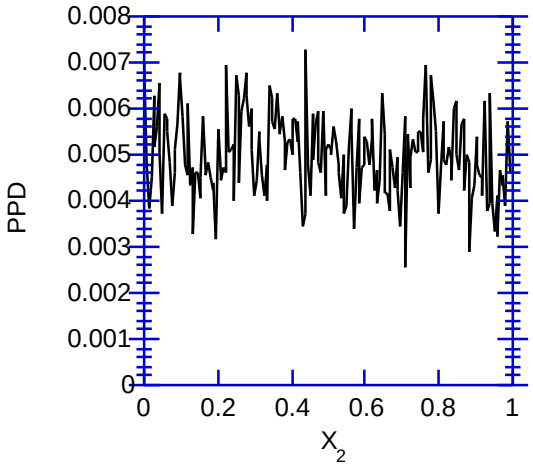


c) X_1, Gen: 10, p_m=1 in 10; d) X_2, Gen: 10, p_m=1 in 10;

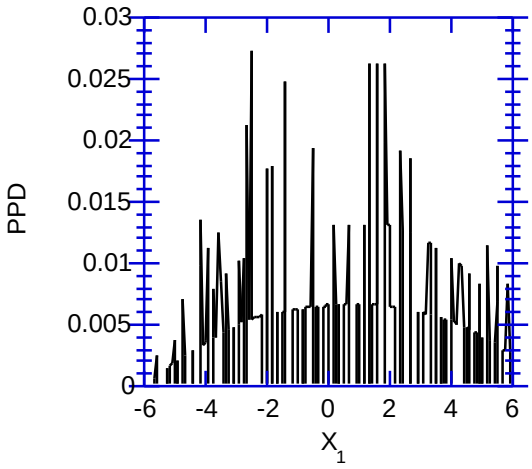
Figure 4. The PPD's for X_1 and X_2 at generation 0, 10, and 100 for different mutation rates.



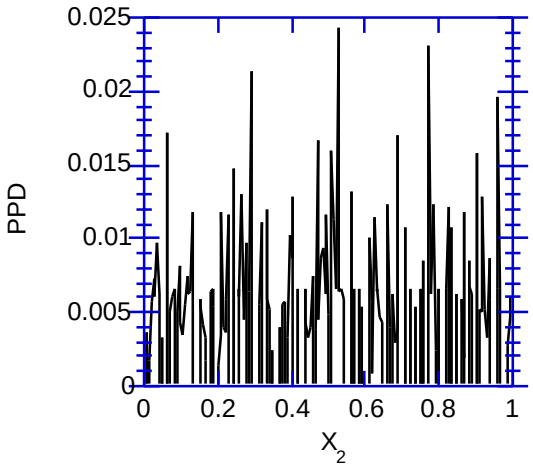
e) X_1, Gen: 100, p_m=1 in 10;



f) X_2, Gen: 100, p_m=1 in 10;

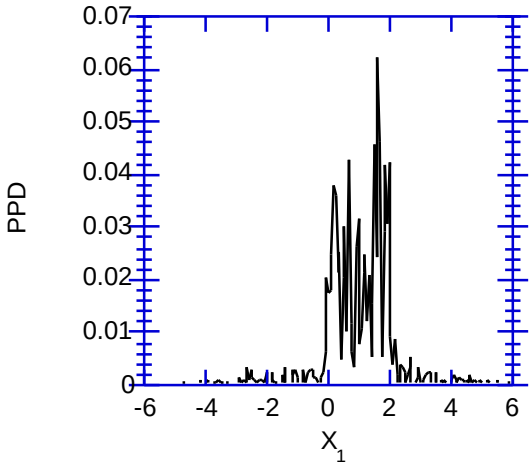


g) X_1, Gen: 0, p_m=1 in 100;

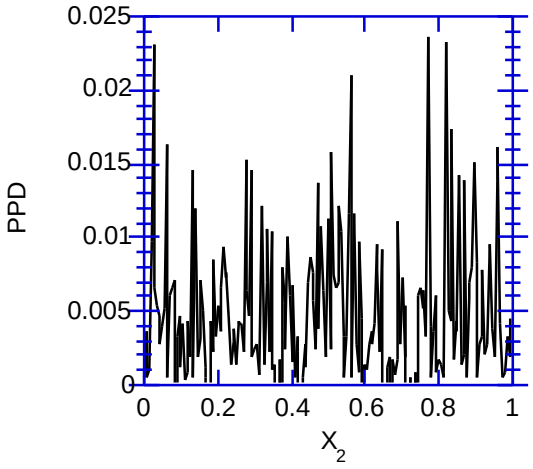


h) X_2, Gen: 0, p_m=1 in 100;

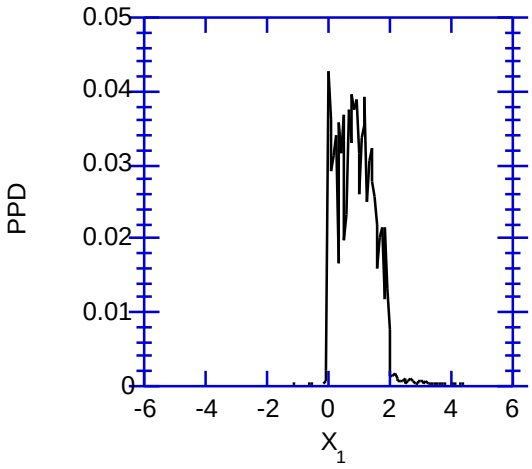
Figure 4. Continued.



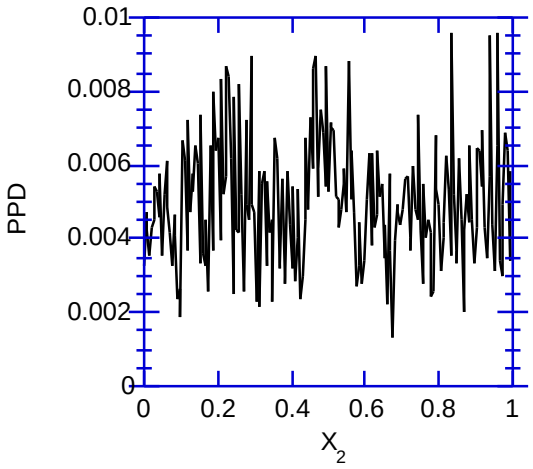
i) X_1, Gen: 10, p_m=1 in 100;



j) X_2, Gen: 10, p_m=1 in 100;

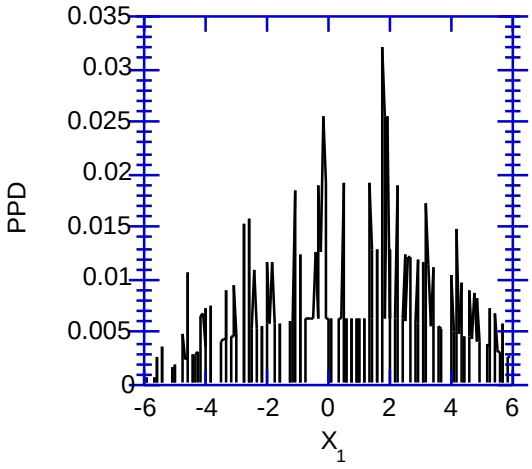


k) X_1, Gen: 100, p_m=1 in 100;

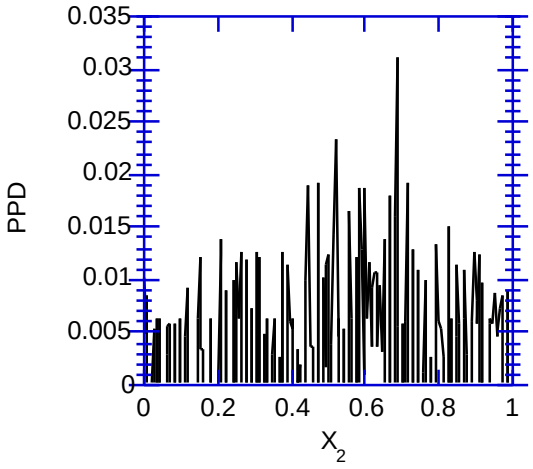


l) X_2, Gen:100, p_m=1 in 100;

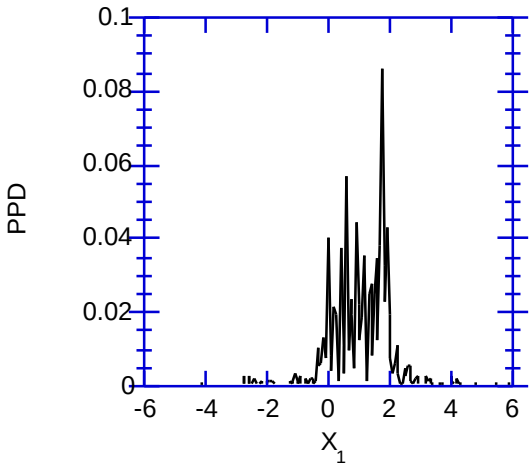
Figure 4. Continued.



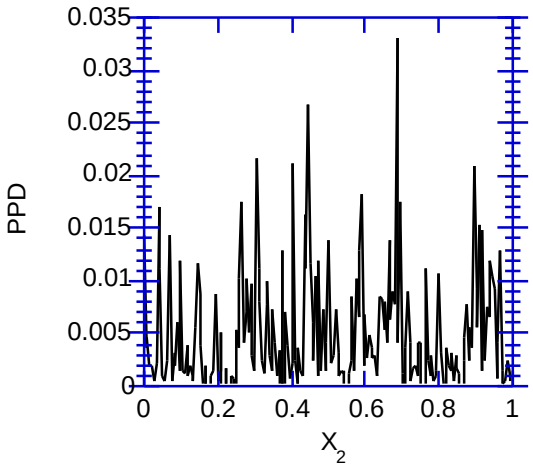
m) X_1, Gen: 0, p_m=1 in 1000;



n) X_2, Gen:0, p_m=1 in 1000;

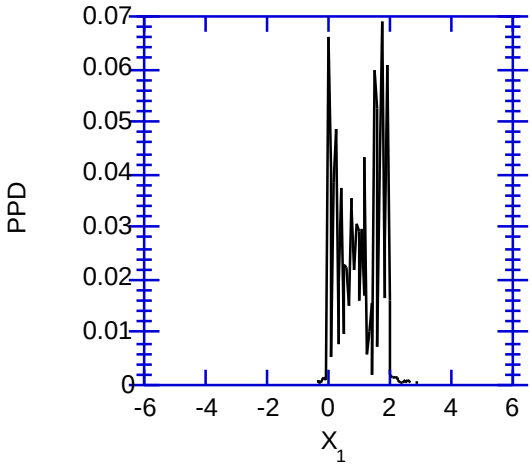


o) X_1, Gen: 10, p_m=1 in 1000;

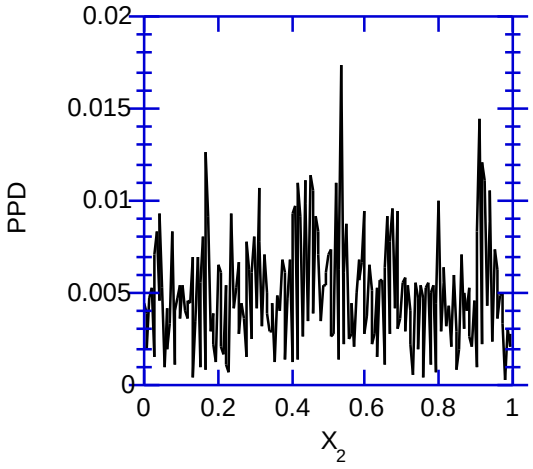


p) X_2, Gen:10, p_m=1 in 1000;

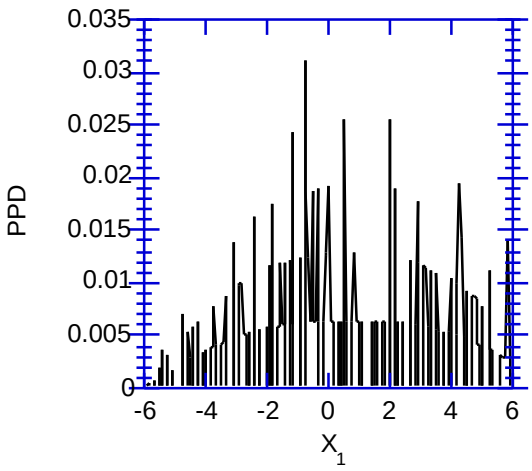
Figure 4. Continued.



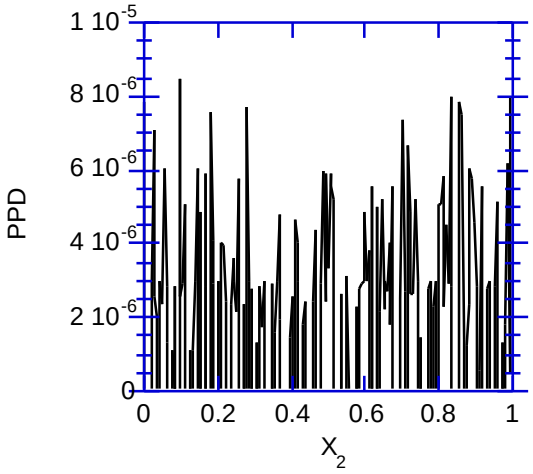
q) X_1, Gen: 100, p_m=1 in 1000;



r) X_2, Gen:100, p_m=1 in 1000;

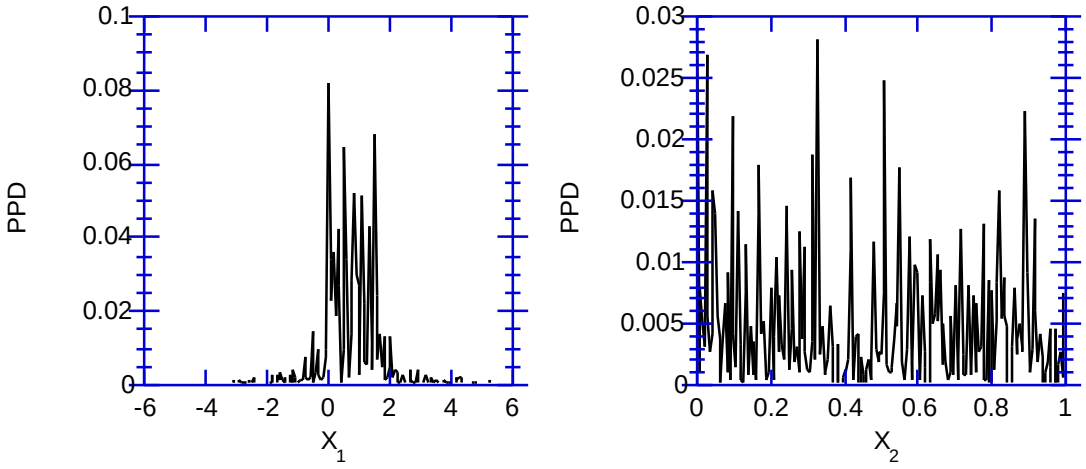


s) X_1, Gen: 0, p_m=1 in 10000;

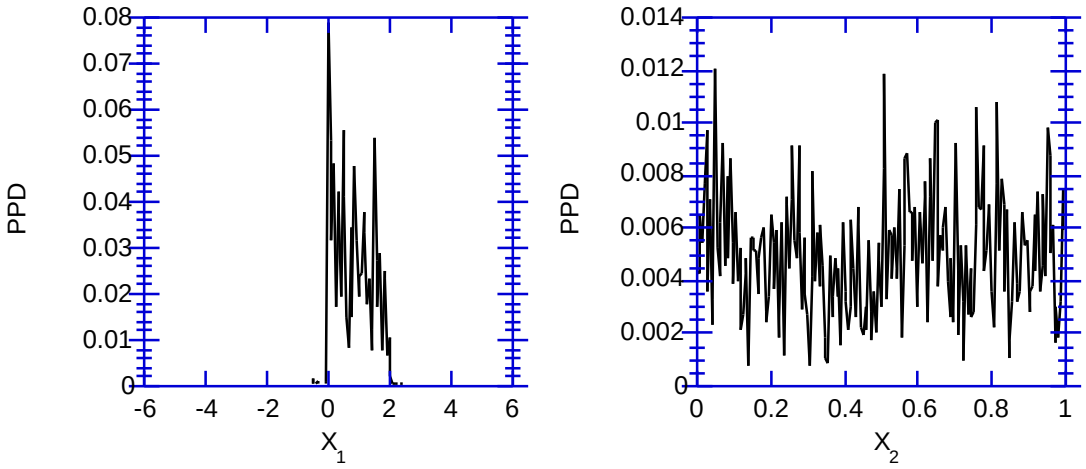


t) X_2, Gen:0, p_m=1 in 10000;

Figure 4. Continued.

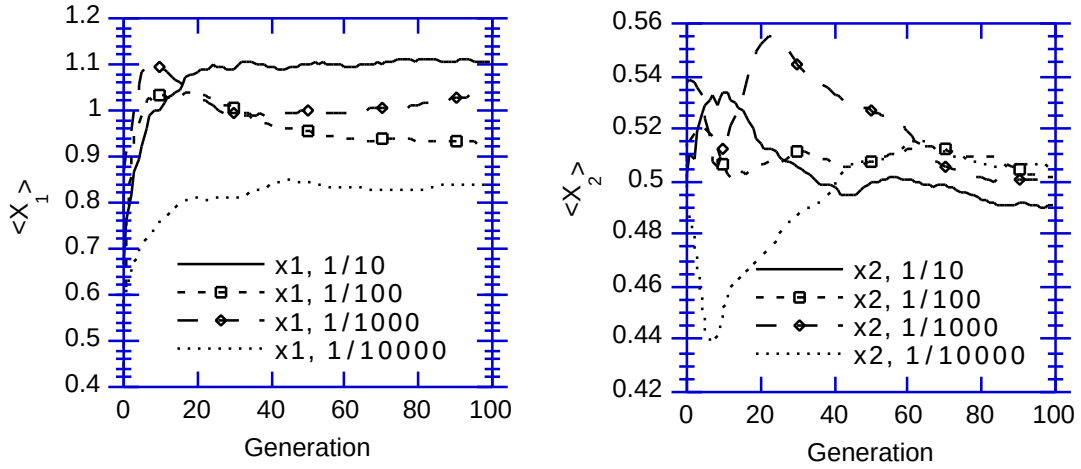


u) X_1, Gen: 10, p_m=1 in 10000; v) X_2, Gen:10, p_m=1 in 10000;



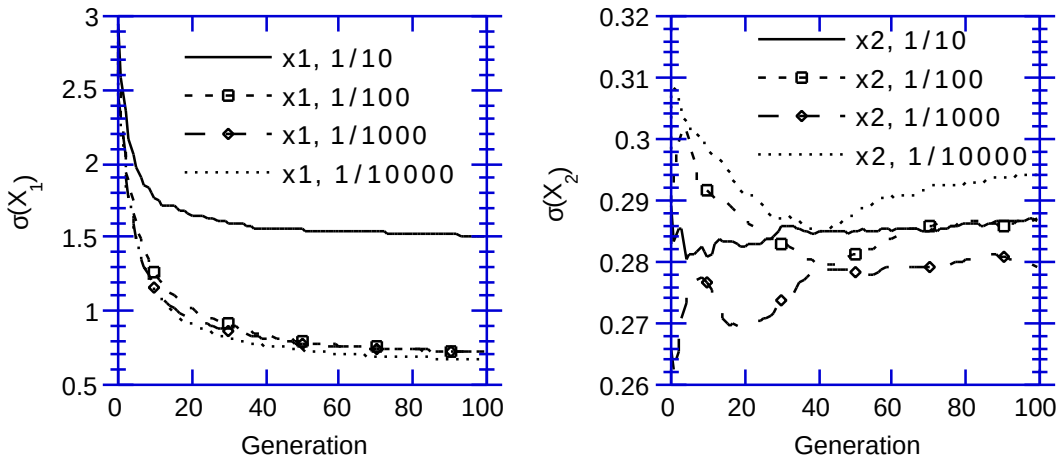
w) X_1, Gen: 100, p_m=1 in 10000; x) X_2, Gen:100, p_m=1 in 10000;

Figure 4. Continued.



a b

Figure 5. The <M> for X_1 and X_2 as a function of generation for different mutation rates. a) X_1; b) X_2;



a b

Figure 6. The standard deviation of the marginal mean models of figure 4. for X_1 and X_2 as a function of generation for different mutation rates. a) X_1; b) X_2;

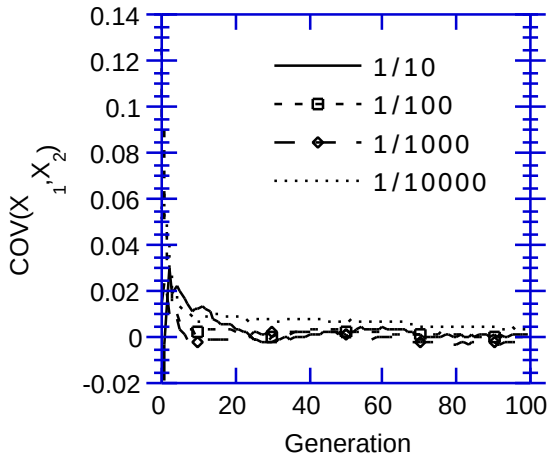


Figure 7. The covariance of X_1 and X_2 as a function of generation for different mutation rates.

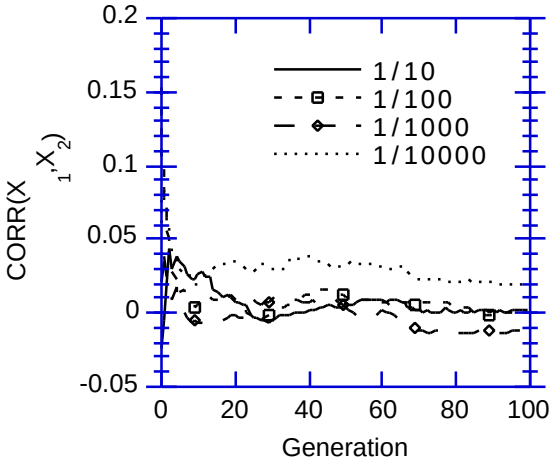
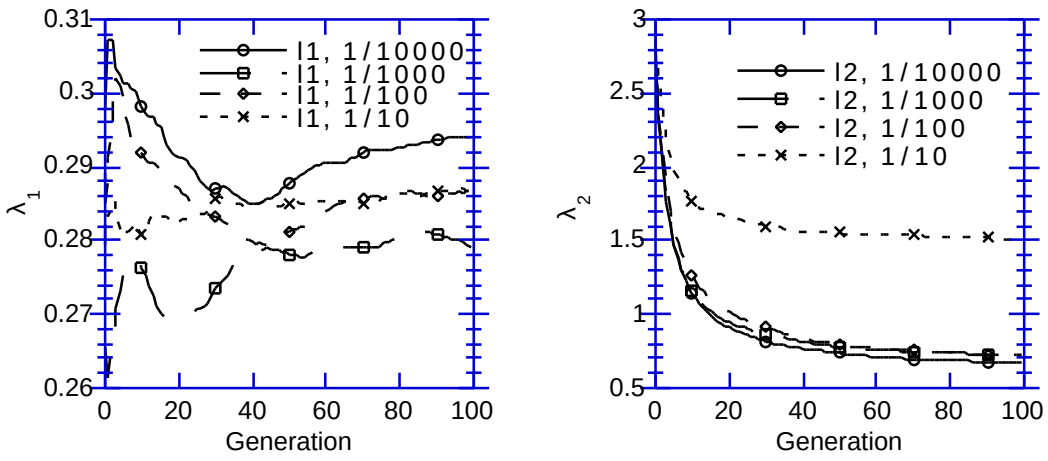


Figure 8. The correlation coefficient of X_1 and X_2 as a function of generation for different mutation rates.



a b

Figure 9. The eigenvalues of the posterior covariance matrix as a function of generation for different mutation rates. a) λ_1; b) λ_2;

Appendix A: The modified Genetic Algorithm Selection code

Appendix A.1: FORTRAN 77 Subroutine FITNESSPROB

```
************************************************************************
************************************************************************
      SUBROUTINE FITNESSPROB()
************************************************************************
************************************************************************
c     THIS SUBROUTINE DETERMINES THE FUZZY FITNESS OF EACH MEMBER
C     OF THE POPULATION AND SAVES THIS DATA IN PROBARRAY
************************************************************************

      INCLUDE 'PARAMSGA'
      INCLUDE 'PARAMSOBJ'
      INCLUDE 'COMMONSOBJ'
      INCLUDE 'COMMONSGA'

      INTEGER I,II
      REAL AVERAGE,X1,X2,X3,X4,X5,Y1,Y2,Y3,Y4,Y5
      REAL TOP,BOTTOM

      DIMENSION RMAX(MXNFNCT),RMIN(MXNFNCT)

      TOP = 1.0
      BOTTOM = 0.0

ccc find the most unfit members of the population for the fuzzy logic selectionmethod
      DO 40 I=1,NFNCT
            RMAX(I) = FNCT(I)+ERROR(I)
            RMIN(I) = FNCT(I)-ERROR(I)
 40   CONTINUE
      DO 41 I = 1,IPOPU
      DO 41 II = 1,NFNCT
            IF (ATT1(I,II).GT.RMAX(II)) RMAX(II) = ATT1(I,II)
            IF (ATT1(I,II).LT.RMIN(II)) RMIN(II) = ATT1(I,II)
 41   CONTINUE

ccC For all objectives or functions determine the fuzzy fitness
      DO 45 I = 1,IPOPU
            AVERAGE = 0.
            DO 35 II = 1,NFNCT
                  X1 = RMIN(II)
            Y1 = BOTTOM
            X2 = FNCT(II)-ERROR(II)
            Y2 = TOP
            X3 = FNCT(II)
            Y3 = TOP
            X4 = FNCT(II)+ERROR(II)
            Y4 = TOP
            X5 = RMAX(II)
            Y5 = BOTTOM
            TEMP = AVG(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5,ATT1(I,II))
                  IF (TEMP.LE.1.0E-6) TEMP = 0.0
                  AVERAGE = AVERAGE+TEMP
 35         CONTINUE
            PROBARRAY(I) = AVERAGE/REAL(NFNCT)
 45         CONTINUE

      RETURN

      END
```

Appendix A.2: FORTRAN 77 Function AVG

```
**************************************************************************
**************************************************************************
     REAL FUNCTION AVG(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5,X)
**************************************************************************
**************************************************************************
ccc  This function is the heart of the fuzzy logic selection method.
ccc  The points listed above define the five major points in a
ccc  fuzzy logic rule set:
ccc  1\          /5
ccc    \        /
ccc     \      /
ccc      2__3__4
ccc AVG determines between which two points the value of X lies
ccc and then calls POINT to determine the cooresponding value of Y
ccccccccccccccccccccccccccccccccccccccccccccccccccccc
              REAL X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5,X,POINT

              IF (X1.GT.X2) THEN
                        PRINT*,'ERROR,X1,X2'
                        STOP
              ENDIF
              IF (X2.GT.X3) THEN
                        PRINT*,'ERROR,X2,X3'
                        STOP
              ENDIF
              IF (X3.GT.X4) THEN
                        PRINT*,'ERROR,X3,X4'
                        STOP
              ENDIF
              IF (X4.GT.X5) THEN
                        PRINT*,'ERROR,X4,X5'
                        STOP
              ENDIF

              IF (X.LE.X2) THEN
                        AVG = POINT(X1,Y1,X2,Y2,X)
              ELSEIF (X.LE.X3) THEN
                        AVG = POINT(X2,Y2,X3,Y3,X)
              ELSEIF (X.LE.X4) THEN
                        AVG = POINT(X3,Y3,X4,Y4,X)
              ELSEIF (X.LE.X5) THEN
                        AVG = POINT(X4,Y4,X5,Y5,X)
              ELSE
                        PRINT*,'ERROR,OUT OF RANGE'
                        STOP
              ENDIF

              RETURN
              END
```

Appendix A.3: FORTRAN 77 Function POINT

```
**************************************************************************
**************************************************************************
              REAL FUNCTION POINT(X1,Y1,X2,Y2,X)
**************************************************************************
**************************************************************************
ccc The function point sends back a value of Y given two points on a line and a value of X.
**************************************************************************

              REAL X1,Y1,X2,Y2,X,SLOPE,B

              IF (ABS(X1-X2).LE.1.0E-6) THEN
                        POINT = Y1
```

```
      ELSE
            SLOPE = (Y2-Y1)/(X2-X1)
            B  = Y1-SLOPE*X1
            POINT = SLOPE*X+B
      ENDIF

      RETURN
      END
```

Appendix A.4: FORTRAN 77 Subroutine REPLI

```
**************************************************************************
**************************************************************************
    SUBROUTINE REPLI(MAX,MEM)
**************************************************************************
**************************************************************************
C  THIS SUBROUTINE RETURNS A MEMBER OF THE POPULATION (MEM) BASED ON ITS
C  FITNESS AND NICHE CROWDING. THE PROCEDURE INVOLVES FIRST FINDING TWO
C  FIT MEMBERS AND THEN SELECTING THE LEAST CROWDED.
**************************************************************************

      INCLUDE 'PARAMSGA'
      INCLUDE 'PARAMSOBJ'
      INCLUDE 'COMMONSOBJ'
      INCLUDE 'COMMONSGA'

      INTEGER M,N,MEM,MAX
      REAL MFIT,NFIT

C     SELECT TO MEMBERS AND GET THEIR FUZZY FITNESS
      CALL PROBSELECT(M,MFIT)
      CALL PROBSELECT(N,NFIT)

      IF (ABS(MFIT-NFIT).LE.0.001) THEN
c      count up the crowding factor for M and N IF
C      M AND N HAVE SIMILAR FITNESSES
            RN1 =  PHCOUNTER(M,MAX)
            RN2 =  PHCOUNTER(N,MAX)

            IF (RN1.LT.RN2) THEN
                  MEM=M
            ELSEIF (RN2.LT.RN1) THEN
                  MEM=N
            ELSE
                  IF (URAND(ISEED).LE.0.5) THEN
                        MEM=M
                  ELSE
                        MEM=N
                  ENDIF
            ENDIF
      ELSE IF (MFIT.GT.NFIT) THEN
            MEM = M
      ELSE IF (NFIT.GT.MFIT) THEN
            MEM = N
      END IF


      RETURN

      END
```

Appendix A.4: FORTRAN 77 Subroutine PROBSELECT

```
**************************************************************************
**************************************************************************
      SUBROUTINE PROBSELECT(N,FIT)
```

```
**************************************************************************
**************************************************************************
C           THIS ROUTINE SELECTS A FIT MEMBER OF THE POPULATION AND RETURNS
C      ITS FUZZY FITNESS
**************************************************************************

      INCLUDE 'PARAMSGA'
      INCLUDE 'PARAMSOBJ'
      INCLUDE 'COMMONSOBJ'
      INCLUDE 'COMMONSGA'

            INTEGER N
            REAL R,FIT

            ICOUNT = 0

 35         ICOUNT = ICOUNT+1
            IF (ICOUNT.GE.10) PRINT*,'ICOUNT=',ICOUNT

C            RANDOMLY SELECT A MEMBER FROM THE POPULATION
            N = NINT(URAND(ISEED)*IPOPU+0.5)

C                     GET A RANDOM NUMBER BETWEEN 0 AND 1
            R = URAND(ISEED)
            IF ((R.LT.0.).OR.(R.GT.1.)) THEN
                        PRINT*, 'MAJOR ERROR IN URAND',R
                        READ(*,*)
                        STOP
            END IF

C                     GET FITNESS OF N WHICH MUST LAY BETWEEN 0 AND 1
            FIT = PROBARRAY(N)
            IF ((FIT.LT.0.).OR.(FIT.GT.1.)) THEN
                        PRINT*, 'MAJOR ERROR IN PROBARRAY',FIT
                        READ(*,*)
                        STOP
            END IF

C                     IF R IS LESS THAN OR EQUAL TO FIT THEN
C            N IS THE MEMBER SELECTED
C                     THIS IS ESSENTIALLY A MONTE CARLO SELECTION SCHEME
C                     IN WHICH THE PROBABILTIY DISTRIBUTION IS DEFINED BY
C                     THE FUZZY RULE SET.
C
            IF (R.LE.FIT) THEN
            ELSE
                        GO TO 35
            END IF

            RETURN
            END
```

Appendix B: The Bayesian Inference Engine (BIE)

Appendix B.1: Program Bayes

```
!PROGRAM BAYES-------------------------------------------------------------
! THIS PROGRAM CALCULATES THE POSTERIOR PROBABILITY DENSITY (PPD),
! MEAN MODEL <M>, AND THE MEAN MODEL COVARIENCE CMN, AS A FUNCTION
! OF GENERATION NUMBER FROM THE OUTPUT OF A GENETIC ALGORITHM
! OPTIMIZATION ROUTINE
!
! VARIABLES:
!           STATUS                        I/O STATUS INDICATOR
```

```
!           I,II,III                DO LOOP COUNTERS
!           GENERATIONS             NUMBER OF GENERATIONS
!           POPULATION_SIZE                 POPULATION SIZE
!           NUMBER_VARIABLES        NUMBER OF VARIABLES
!           NUMBER_OBJECTIVES       NUMBER OF OBJECTIVES
!           NUMBER_BIN = 200        NUMBER OF BINS (CONSTANT)
!           TEMP                    JUNK VARIABLE
!           VARIABLE_RANGE                  RANGE OF VARIABLES
!           EXPERIMENTAL_DATA       EXPERIMENTAL DATA
!           EXPERIMENTAL_ERROR      EXPERIMENTAL DATA ERROR
!           EXPECTATION             AVERAGE VALUE FROM PPD
!           MODELVALUE              REAL VALUES OF BINS
!
! INPUT:
!           FILE='HEADGA'           GA PARAMETERS
!           FILE='HEADVAR'          VARIABLE LIMITS
!           FILE='HEADOBJ'          OBJECTIVE DATA
!
! OUTPUT:
!           NONE
!
! SUBROUTINES:
!           PPD_GENERATOR
!           OPTIMUM_MODEL
!           MODEL_VALUE
!
! FUNCTIONS:
!           NONE
!-------------------------------------------------------------------------
PROGRAM BAYES

IMPLICIT NONE

INTEGER :: STATUS,I,II,III

INTEGER :: GENERATIONS,POPULATION_SIZE
INTEGER :: NUMBER_VARIABLES
INTEGER :: NUMBER_OBJECTIVES
INTEGER, PARAMETER :: NUMBER_BIN = 200

REAL :: TEMP
REAL, ALLOCATABLE :: VARIABLE_RANGE(:,:)
REAL, ALLOCATABLE :: EXPERIMENTAL_DATA(:),EXPERIMENTAL_ERROR(:),MODELVALUE(:,:)

INTERFACE
        SUBROUTINE
PPD_GENERATOR(GENERATIONS,POPULATION_SIZE,NUMBER_VARIABLES,NUMBER_OBJECTIVES,VARIABLE_RANGE,&
                EXPERIMENTAL_DATA,EXPERIMENTAL_ERROR,NUMBER_BIN)
                IMPLICIT NONE
                INTEGER :: I,II,IIISTATUS
                INTEGER,INTENT(IN) :: GENERATIONS,POPULATION_SIZE
                INTEGER,INTENT(IN) :: NUMBER_VARIABLES
                INTEGER,INTENT(IN) :: NUMBER_OBJECTIVES
                INTEGER,INTENT(IN) :: NUMBER_BIN
                REAL,INTENT(IN) :: VARIABLE_RANGE(NUMBER_VARIABLES,2)
                REAL,INTENT(IN) ::
EXPERIMENTAL_DATA(NUMBER_OBJECTIVES),EXPERIMENTAL_ERROR(NUMBER_OBJECTIVES)
                REAL, ALLOCATABLE :: MODEL_VAR(:,:),CALC_DATA(:,:),PM(:),PPD(:,:)
        END SUBROUTINE PPD_GENERATOR

        SUBROUTINE
OPTIMUM_MODEL(GENERATIONS,NUMBER_VARIABLES,NUMBER_BIN,POPULATION_SIZE,MODELVALUE)
                IMPLICIT NONE
                INTEGER :: I, II, STATUS, SEED, N
                INTEGER, INTENT(IN) :: GENERATIONS,NUMBER_VARIABLES,NUMBER_BIN,POPULATION_SIZE
                REAL, INTENT(IN) :: MODELVALUE(NUMBER_BIN,NUMBER_VARIABLES)
```

```
                REAL, ALLOCATABLE :: PPD(:,:),OP_MOD(:,:)
                LOGICAL :: SELECTED
        END SUBROUTINE OPTIMUM_MODEL

        SUBROUTINE MODEL_VALUE(NUMBER_VARIABLES,NUMBER_BIN,VARIABLE_RANGE,MODELVALUE)
                IMPLICIT NONE
                INTEGER :: I,II
                INTEGER, INTENT(IN) :: NUMBER_VARIABLES,NUMBER_BIN
                REAL, INTENT(IN) :: VARIABLE_RANGE(NUMBER_VARIABLES,2)
                REAL, INTENT(OUT) :: MODELVALUE(NUMBER_BIN,NUMBER_VARIABLES)
        END SUBROUTINE MODEL_VALUE
END INTERFACE

OPEN (UNIT=10,FILE='HEADGA',STATUS='OLD',ACTION='READ',POSITION='REWIND',IOSTAT=STATUS)
        IF (STATUS>0) STOP "CAN NOT OPEN FILE: HEADGA"

        READ(UNIT=10,FMT=*,ERR=20,IOSTAT=STATUS)
        READ(UNIT=10,FMT=*,ERR=20,IOSTAT=STATUS) GENERATIONS
        READ(UNIT=10,FMT=*,ERR=20,IOSTAT=STATUS) POPULATION_SIZE
CLOSE(UNIT=10)

OPEN (UNIT=10,FILE='HEADVAR',STATUS='OLD',ACTION='READ',POSITION='REWIND',IOSTAT=STATUS)
        IF (STATUS>0) STOP "CAN NOT OPEN FILE: HEADVAR"

        READ(UNIT=10,FMT=*,ERR=20,IOSTAT=STATUS) NUMBER_VARIABLES
        ALLOCATE(VARIABLE_RANGE(NUMBER_VARIABLES,2),STAT=STATUS)
                IF (STATUS /= 0) STOP "NOT ENOUGH MEMORY FOR VARIABLE_RANGE"

        VAR_READER: DO I = 1,NUMBER_VARIABLES,1
                READ(UNIT=10,FMT=*,ERR=20,IOSTAT=STATUS) (VARIABLE_RANGE(I,II),II=1,2,1)
        END DO VAR_READER
CLOSE(UNIT=10)

OPEN (UNIT=10,FILE='HEADOBJ',STATUS='OLD',ACTION='READ',POSITION='REWIND',IOSTAT=STATUS)
        IF (STATUS>0) STOP "CAN NOT OPEN FILE: HEADOBJ"

        READ(UNIT=10,FMT=*,ERR=20,IOSTAT=STATUS) NUMBER_OBJECTIVES
        ALLOCATE(EXPERIMENTAL_DATA(NUMBER_OBJECTIVES),EXPERIMENTAL_ERROR(NUMBER_OBJECTIVES),ST
AT=STATUS)
                IF (STATUS /= 0) STOP "NOT ENOUGH MEMORY FOR EXPERIMENTAL_DATA AND/OR
EXPERIMENTAL_ERROR"

        EXP_READER: DO I = 1,NUMBER_OBJECTIVES,1
                READ(UNIT=10,FMT=*,ERR=20,IOSTAT=STATUS) EXPERIMENTAL_DATA(I),EXPERIMENTAL_ERROR(I)
        END DO EXP_READER
20 IF (STATUS > 0) PRINT*,'INPUT DATA ERROR'
CLOSE(UNIT=10)

CALL
PPD_GENERATOR(GENERATIONS,POPULATION_SIZE,NUMBER_VARIABLES,NUMBER_OBJECTIVES,VARIABLE_RANGE,&
EXPERIMENTAL_DATA,EXPERIMENTAL_ERROR,NUMBER_BIN)

ALLOCATE(MODELVALUE(NUMBER_BIN,NUMBER_VARIABLES),STAT=STATUS)
        IF (STATUS /= 0) STOP "NOT ENOUGH MEMORY FOR MODELVALUE"

CALL MODEL_VALUE(NUMBER_VARIABLES,NUMBER_BIN,VARIABLE_RANGE,MODELVALUE)

CALL OPTIMUM_MODEL(GENERATIONS,NUMBER_VARIABLES,NUMBER_BIN,POPULATION_SIZE,MODELVALUE)

DEALLOCATE(MODELVALUE,STAT=STATUS)
        IF (STATUS /= 0) STOP "DEALLOCATE ERROR FOR MODELVALUE"

END PROGRAM BAYES
```

Appendix B.2: SUBROUTINE PPD_GENERATOR

```
!PPD_GENERATOR-------------------------------------------------------------
!THIS ROUTINE GENERATES THE POSTERIOR PROBABILITY DENSITY AND SAVES
!IT IN A FILE CALLED PPD.  THE FILE CONTAINS THE PPD FOR EACH MODEL VARIABLE
!FOR EACH GENERATION STARTING WITH GENERATION 0.
!
! VARIABLES:
!           I,II,III                DO LOOP COUNTERS
!           STATUS                  I/O STATUS INDICATOR
!           BIN_NUMBER              BIN NUMBER FROM BINNER
!           GENERATIONS             NUMBER OF GENERATIONS
!           POPULATION_SIZE         POPULATION SIZE
!           NUMBER_VARIABLES        NUMBER OF VARIABLES
!           NUMBER_OBJECTIVES       NUMBER OF OBJECTIVES
!           NUMBER_BIN              NUMBER OF BINS
!           VARIABLE_RANGE          ARRAY OF MIN AND MAX OF VARIABLES
!           EXPERIMENTAL_DATA       EXPERIMENTAL DATA
!           EXPERIMENTAL_ERROR      EXPERIMENTAL DATA ERROR
!           MODEL_VAR               MODEL VARIABLES
!           CALC_DATA               CALCULATED DATA
!           PM                      PROBABILITY DISTRIBUTIONS
!           PPD                     POSTIERIOR PROBABILITY DENSITY
!           SUMTOTAL                SUM OF ALL SCALED FITNESS
! INPUT:
!           GENERATIONS
!           POPULATION_SIZE
!           NUMBER_VARIABLES
!           NUMBER_OBJECTIVES
!           VARIABLE_RANGE
!           EXPERIMENTAL_DATA
!           EXPERIMENTAL_ERROR
!           TAPE7                   CALCULATED DATA
!           TAPE8                   MODEL DATA
! OUTPUT:
!           PPD                     POSTERIOR PROBABILITY DENSITY FILE
! SUBROUTINES:
!           NORMAL
! FUNCTIONS:
!           BINNER
!-------------------------------------------------------------------------

SUBROUTINE
PPD_GENERATOR(GENERATIONS,POPULATION_SIZE,NUMBER_VARIABLES,NUMBER_OBJECTIVES,VARIABLE_RANGE,&
EXPERIMENTAL_DATA,EXPERIMENTAL_ERROR,NUMBER_BIN)

IMPLICIT NONE

INTEGER :: I,II,III,STATUS,IERR,BIN_NUMBER
INTEGER,INTENT(IN) :: GENERATIONS,POPULATION_SIZE
INTEGER,INTENT(IN) :: NUMBER_VARIABLES
INTEGER,INTENT(IN) :: NUMBER_OBJECTIVES
INTEGER,INTENT(IN) :: NUMBER_BIN

REAL,INTENT(IN) :: VARIABLE_RANGE(NUMBER_VARIABLES,2)
REAL,INTENT(IN) :: EXPERIMENTAL_DATA(NUMBER_OBJECTIVES),EXPERIMENTAL_ERROR(NUMBER_OBJECTIVES)
REAL, ALLOCATABLE :: MODEL_VAR(:,:),CALC_DATA(:,:),PM(:),PPD(:,:),AVERAGEMODEL(:),PROD1(:,:),&
                     COVAR(:,:),CORR(:,:),COVARTEMP(:,:),W(:),Z(:,:),FV1(:)
REAL :: SUMTOTAL,A,B,C,X1,X2
!-------------------------------------------------------------------------

INTERFACE
        FUNCTION BINNER(VARID,VAR,NUMBER_BIN,NUMBER_VARIABLES,VARIABLE_RANGE)
                IMPLICIT NONE
                INTEGER :: BINNER
                INTEGER, INTENT(IN) :: VARID,NUMBER_BIN,NUMBER_VARIABLES
                REAL, INTENT(IN) :: VAR,VARIABLE_RANGE(NUMBER_VARIABLES,2)
        END FUNCTION BINNER

        SUBROUTINE NORMAL(POPULATION_SIZE,NUMBER_OBJECTIVES,EXPERIMENTAL_DATA,&
```

```
                    EXPERIMENTAL_ERROR,CALC_DATA,PM)
                    IMPLICIT NONE
                    INTEGER :: I,II
                    INTEGER, INTENT(IN) :: POPULATION_SIZE
                    INTEGER, INTENT(IN) :: NUMBER_OBJECTIVES
                    REAL, INTENT(IN) ::
EXPERIMENTAL_DATA(NUMBER_OBJECTIVES),EXPERIMENTAL_ERROR(NUMBER_OBJECTIVES)
                    REAL, INTENT(IN) :: CALC_DATA(POPULATION_SIZE,NUMBER_OBJECTIVES)
                    REAL, INTENT(OUT) :: PM(POPULATION_SIZE)
                    REAL, PARAMETER :: SCALER = 1.0
                    REAL, ALLOCATABLE :: RMAX(:),RMIN(:)
                    REAL :: X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5,X,AVERAGE
          END SUBROUTINE NORMAL
END INTERFACE

!-----------------------------------------------------------------------
ALLOCATE(MODEL_VAR(POPULATION_SIZE,NUMBER_VARIABLES),STAT=STATUS)
          IF (STATUS /= 0) STOP "NOT ENOUGH MEMORY FOR MODEL_VAR"
ALLOCATE(CALC_DATA(POPULATION_SIZE,NUMBER_OBJECTIVES),STAT=STATUS)
          IF (STATUS /= 0) STOP "NOT ENOUGH MEMORY FOR CALC_DATA"
ALLOCATE(PM(POPULATION_SIZE),STAT=STATUS)
          IF (STATUS /= 0) STOP "NOT ENOUGH MEMORY FOR PM"
ALLOCATE(PPD(NUMBER_VARIABLES,NUMBER_BIN),STAT=STATUS)
          IF (STATUS /= 0) STOP "NOT ENOUGH MEMORY FOR PPD"
!-----------------------------------------------------------------------
ALLOCATE(AVERAGEMODEL(NUMBER_VARIABLES),STAT=STATUS)
          IF (STATUS /= 0) STOP "NOT ENOUGH MEMORY FOR AVERAGEMODEL"
ALLOCATE(COVAR(NUMBER_VARIABLES,NUMBER_VARIABLES),STAT=STATUS)
          IF (STATUS /= 0) STOP "NOT ENOUGH MEMORY FOR COVAR"
ALLOCATE(CORR(NUMBER_VARIABLES,NUMBER_VARIABLES),STAT=STATUS)
          IF (STATUS /= 0) STOP "NOT ENOUGH MEMORY FOR CORR"
!-----------------------------------------------------------------------
ALLOCATE(PROD1(NUMBER_VARIABLES,NUMBER_VARIABLES),STAT=STATUS)
          IF (STATUS /= 0) STOP "NOT ENOUGH MEMORY FOR PROD1"
ALLOCATE(COVARTEMP(NUMBER_VARIABLES,NUMBER_VARIABLES),STAT=STATUS)
          IF (STATUS /= 0) STOP "NOT ENOUGH MEMORY FOR COVARTEMP"
!-----------------------------------------------------------------------
ALLOCATE(W(NUMBER_VARIABLES),STAT=STATUS)
          IF (STATUS /= 0) STOP "NOT ENOUGH MEMORY FOR W"
ALLOCATE(Z(NUMBER_VARIABLES,NUMBER_VARIABLES),STAT=STATUS)
          IF (STATUS /= 0) STOP "NOT ENOUGH MEMORY FOR Z"
ALLOCATE(FV1(NUMBER_VARIABLES),STAT=STATUS)
          IF (STATUS /= 0) STOP "NOT ENOUGH MEMORY FOR FV1"
!-----------------------------------------------------------------------

!-----------------------------------------------------------------------
! INPUT FILES
!-----------------------------------------------------------------------
OPEN (UNIT=10,FILE='TAPE7',STATUS='OLD',ACTION='READ',POSITION='REWIND',IOSTAT=STATUS)
          IF (STATUS>0) STOP "CAN NOT OPEN FILE: TAPE7"
OPEN (UNIT=20,FILE='TAPE8',STATUS='OLD',ACTION='READ',POSITION='REWIND',IOSTAT=STATUS)
          IF (STATUS>0) STOP "CAN NOT OPEN FILE: TAPE8"
!-----------------------------------------------------------------------
! OUTPUT FILES
!-----------------------------------------------------------------------
OPEN (UNIT=30,FILE='PPD',STATUS='REPLACE',ACTION='WRITE',POSITION='REWIND',IOSTAT=STATUS)
          IF (STATUS>0) STOP "CAN NOT OPEN FILE: PPD"
OPEN (UNIT=40,FILE='EXPECTATION',STATUS='REPLACE',ACTION='WRITE',POSITION='REWIND',IOSTAT=STATUS)
          IF (STATUS>0) STOP "CAN NOT OPEN FILE: AVERAGETEMP"
OPEN (UNIT=50,FILE='COVARIENCE',STATUS='REPLACE',ACTION='WRITE',POSITION='REWIND',IOSTAT=STATUS)
          IF (STATUS>0) STOP "CAN NOT OPEN FILE: COVARITEMP"
OPEN (UNIT=60,FILE='CORRELATION',STATUS='REPLACE',ACTION='WRITE',POSITION='REWIND',IOSTAT=STATUS)
          IF (STATUS>0) STOP "CAN NOT OPEN FILE: CORRELATETEMP"
OPEN (UNIT=70,FILE='STANDARD_DEV',STATUS='REPLACE',ACTION='WRITE',POSITION='REWIND',IOSTAT=STATUS)
          IF (STATUS>0) STOP "CAN NOT OPEN FILE: STANDARD_DEV"
OPEN (UNIT=80,FILE='EIGENVALUES',STATUS='REPLACE',ACTION='WRITE',POSITION='REWIND',IOSTAT=STATUS)
          IF (STATUS>0) STOP "CAN NOT OPEN FILE: EIGENVALUES"
OPEN (UNIT=90,FILE='EIGENVECTORS',STATUS='REPLACE',ACTION='WRITE',POSITION='REWIND',IOSTAT=STATUS)
          IF (STATUS>0) STOP "CAN NOT OPEN FILE: EIGENVECTORS"
```

```
!-------------------------------------------------------------------------

PPD = 0.0
SUMTOTAL = 0.0

GEN_LOOP1: DO I = 0,GENERATIONS
        CALC_DATA = 0.0
        MODEL_VAR = 0.0

        ! READING IN THE OBJECTIVE VALUES
        POP_LOOP1: DO II = 1,POPULATION_SIZE,1
                READ(UNIT=10,FMT=*,ERR=40,IOSTAT=STATUS) (CALC_DATA(II,III),III=1,NUMBER_OBJECTIVES,1)
        END DO POP_LOOP1
        POP_LOOP4: DO II = 1,POPULATION_SIZE,1
                OBJ_LOOP1: DO III = 1,NUMBER_OBJECTIVES,1
                        CALC_DATA(II,III) = CALC_DATA(II,III) + EXPERIMENTAL_DATA(III)
                END DO OBJ_LOOP1
        END DO POP_LOOP4

        ! READING IN THE VARIABLE VALUES
        POP_LOOP2: DO II = 1,POPULATION_SIZE,1
                READ(UNIT=20,FMT=*,ERR=40,IOSTAT=STATUS) (MODEL_VAR(II,III),III=1,NUMBER_VARIABLES)
        END DO POP_LOOP2

        CALL
NORMAL(POPULATION_SIZE,NUMBER_OBJECTIVES,EXPERIMENTAL_DATA,EXPERIMENTAL_ERROR,CALC_DATA,PM)

        SUMTOTAL = SUMTOTAL + SUM(PM)

        ! BIN THE DATA TO CREATE THE POSTERIOR PROBABILITY DENSITY
        POP_LOOP3: DO II = 1,POPULATION_SIZE,1
                VAR_LOOP1: DO III = 1,NUMBER_VARIABLES,1
                        BIN_NUMBER =
BINNER(III,MODEL_VAR(II,III),NUMBER_BIN,NUMBER_VARIABLES,VARIABLE_RANGE)
                        PPD(III,BIN_NUMBER) = PPD(III,BIN_NUMBER)+PM(II)
                END DO VAR_LOOP1
        END DO POP_LOOP3
        VAR_LOOP2: DO II = 1,NUMBER_VARIABLES,1
                WRITE(UNIT=30,FMT=*,ERR=40,IOSTAT=STATUS) (PPD(II,III),III=1,NUMBER_BIN,1)
        END DO VAR_LOOP2


        ! CALCULATION THE EXPECATION VALUE
        VAR_LOOP3: DO II = 1, NUMBER_VARIABLES, 1
                POP_LOOP5: DO III = 1,POPULATION_SIZE, 1
                        AVERAGEMODEL(II) = AVERAGEMODEL(II) + MODEL_VAR(III,II)*PM(III)
                END DO POP_LOOP5
        END DO VAR_LOOP3
        WRITE(UNIT=40,FMT=*,ERR=40,IOSTAT=STATUS) I,(AVERAGEMODEL(II)/SUMTOTAL,II=1,NUMBER_VARIABLES,1)


        ! CALCULATION OF THE COVARIENCE MATRIX
        POP_LOOP6: DO II = 1,POPULATION_SIZE, 1
                CALL MATMUL1(PROD1,MODEL_VAR(II,1:NUMBER_VARIABLES:1),&
                MODEL_VAR(II,1:NUMBER_VARIABLES:1),NUMBER_VARIABLES)
                COVARTEMP = COVARTEMP + PROD1*PM(II)
        END DO POP_LOOP6
        CALL MATMUL1(PROD1,AVERAGEMODEL/SUMTOTAL,AVERAGEMODEL/SUMTOTAL,NUMBER_VARIABLES)
        COVAR = (COVARTEMP/SUMTOTAL) - PROD1
        WRITE(UNIT=50,FMT='(I4)',ADVANCE='NO',ERR=40,IOSTAT=STATUS) I
        VAR_LOOP4: DO II = 1,NUMBER_VARIABLES,1
                WRITE(UNIT=50,FMT='(20E15.5)',ADVANCE='NO',ERR=40,IOSTAT=STATUS) &
                        (COVAR(II,III),III=II+1,NUMBER_VARIABLES,1)
        END DO VAR_LOOP4
        WRITE(UNIT=50,FMT=*,ERR=40,IOSTAT=STATUS)


        !Calculate Eigenvalues of the Covariance Matrix
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!ccc     The Eigenvalue/Eigenvector subroutines come from:
```

```
!ccc        “Matrix Eigensystems Routines – EISPACK Guide” Second Edition
!ccc        B.T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, C. B. Moler
!ccc        Published in Lecture Notes in Computer Science
!ccc        Edited by G. Goos, J. Hartmanis
!ccc        Springer-Verlag, New York, 1976
! ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
            W = 0.0
            FV1 = 0.0
            Z = 0.0
            CALL TRED2(NUMBER_VARIABLES,NUMBER_VARIABLES,COVAR,W,FV1,Z)
            CALL TQL2(NUMBER_VARIABLES,NUMBER_VARIABLES,W,FV1,Z,IERR)
            IF (IERR /= 0) PRINT*,'PROBLEM WITH EIGENSYSTEM'
            W = W*W
            W = SQRT(SQRT(W))
            WRITE(UNIT=80,FMT=*,ERR=40,IOSTAT=STATUS) I,(W(II),II=1,NUMBER_VARIABLES),SUM(W)
            IF (I==GENERATIONS) THEN
                  VAR_LOOP5: DO II = 1,NUMBER_VARIABLES,1
                        WRITE(UNIT=90,FMT=*,ERR=40,IOSTAT=STATUS) (Z(II,III),III=1,NUMBER_VARIABLES)
                  END DO VAR_LOOP5
            END IF


            !CALCULATION OF THE CORRELATION MATRIX
            VAR_LOOP6:DO II = 1, NUMBER_VARIABLES, 1
                  VAR_LOOP7: DO III = 1,NUMBER_VARIABLES, 1
                        CORR(II,III) = COVAR(II,III)/(SQRT(COVAR(II,II))*SQRT(COVAR(III,III)))
                  END DO VAR_LOOP7
            END DO VAR_LOOP6
            WRITE(UNIT=60,FMT='(I4)',ADVANCE='NO',ERR=40,IOSTAT=STATUS) I
            VAR_LOOP8: DO II = 1,NUMBER_VARIABLES,1
                  WRITE(UNIT=60,FMT='(20E15.5)',ADVANCE='NO',ERR=40,IOSTAT=STATUS) &
                        (CORR(II,III),III=II+1,NUMBER_VARIABLES,1)
            END DO VAR_LOOP8
            WRITE(UNIT=60,FMT=*,ERR=40,IOSTAT=STATUS)

            !OUTPUT OF THE STANDARD DEVIATION
            COVAR = SQRT(COVAR)
            WRITE(UNIT=70,FMT=*,ERR=40,IOSTAT=STATUS) I,(COVAR(III,III),III=1,NUMBER_VARIABLES,1)


END DO GEN_LOOP1
40 IF (STATUS > 0) PRINT*,'INPUT DATA ERROR'

!------------------------------------------------------------------------------
CLOSE(UNIT=90)
CLOSE(UNIT=80)
CLOSE(UNIT=70)
CLOSE(UNIT=60)
CLOSE(UNIT=50)
CLOSE(UNIT=40)
CLOSE(UNIT=30)
CLOSE(UNIT=20)
CLOSE(UNIT=10)
!------------------------------------------------------------------------------

DEALLOCATE(AVERAGEMODEL,STAT=STATUS)
            IF (STATUS /= 0) STOP "DEALLOCATE ERROR FOR AVERAGEMODEL"
DEALLOCATE(PROD1,STAT=STATUS)
            IF (STATUS /= 0) STOP "DEALLOCATE ERROR FOR PROD1"
DEALLOCATE(COVAR,STAT=STATUS)
            IF (STATUS /= 0) STOP "DEALLOCATE ERROR FOR COVAR"
DEALLOCATE(COVARTEMP,STAT=STATUS)
            IF (STATUS /= 0) STOP "DEALLOCATE ERROR FOR COVARTEMP"
DEALLOCATE(CORR,STAT=STATUS)
            IF (STATUS /= 0) STOP "DEALLOCATE ERROR FOR CORR"
DEALLOCATE(CALC_DATA,STAT=STATUS)
            IF (STATUS /= 0) STOP "DEALLOCATE ERROR FOR CALC_DATA"
DEALLOCATE(MODEL_VAR,STAT=STATUS)
            IF (STATUS /= 0) STOP "DEALLOCATE ERROR FOR MODEL_VAR"
DEALLOCATE(PM,STAT=STATUS)
```

```
	IF (STATUS /= 0) STOP "DEALLOCATE ERROR FOR PM"
DEALLOCATE(PPD,STAT=STATUS)
	IF (STATUS /= 0) STOP "DEALLOCATE ERROR FOR PPD"
DEALLOCATE(W,STAT=STATUS)
	IF (STATUS /= 0) STOP "DEALLOCATE ERROR FOR W"
DEALLOCATE(Z,STAT=STATUS)
	IF (STATUS /= 0) STOP "DEALLOCATE ERROR FOR Z"
DEALLOCATE(FV1,STAT=STATUS)
	IF (STATUS /= 0) STOP "DEALLOCATE ERROR FOR FV1"

END SUBROUTINE PPD_GENERATOR
```

Appendix B.3: SUBROUTINE MODEL_VALUE

```
!MODEL_VALUE----------------------------------------------------------------
! THIS SUBROUTINE TAKES AS INPUT THE NUMBER OF VARIABLES, THE NUMBER OF BINS IN THE
! POSTERIOR PROBABILITY DENSTIY (PPD), AND THE VARIABLE RANGE ARRAY FOR EACH VARIABLE
! AND SENDS AS OUTPUT THE ARRAY MODELVALUE WHICH HAS A REAL VALUED ASSIGNMENT GIVEN TO
! EACH BIN
!
! VARIABLES:
!	I,II			DO LOOP COUNTERS
!	NUMBER_VARIABLES	NUMBER OF VARIABLES
!	NUMBER_BIN		NUMBER OF BINS
!	VARIABLE_RANGE			ARRAY CONTAINING VARIABLE RANGE
!	MODELVALUE		MODELVALUE ARRAY FOR EACH BIN
! INPUT:
!	NUMBER_VARIABLES	NUMBER OF VARIABLES
!	NUMBER_BIN		NUMBER OF BINS
!	VARIABLE_RANGE			ARRAY CONTAINING VARIABLE RANGE
! OUTPUT:
!	MODELVALUE		MODELVALUE ARRAY FOR EACH BIN
! SUBROUTINES:
!	NONE
! FUNCTIONS:
!	BINVAL
!-------------------------------------------------------------------------

SUBROUTINE MODEL_VALUE(NUMBER_VARIABLES,NUMBER_BIN,VARIABLE_RANGE,MODELVALUE)

IMPLICIT NONE

INTEGER :: I,II
INTEGER, INTENT(IN) :: NUMBER_VARIABLES,NUMBER_BIN
REAL, INTENT(IN) :: VARIABLE_RANGE(NUMBER_VARIABLES,2)
REAL, INTENT(OUT) :: MODELVALUE(NUMBER_BIN,NUMBER_VARIABLES)

INTERFACE
	FUNCTION BINVAL(VARID,BINID,NUMBER_VARIABLES,NUMBER_BIN,VARIABLE_RANGE)
		IMPLICIT NONE
		INTEGER,INTENT(IN) :: VARID,BINID,NUMBER_VARIABLES,NUMBER_BIN
		REAL :: BINVAL,MAXIMUMVAR,MINIMUMVAR,SLOPE,B,Y1,Y2
		REAL, INTENT(IN) :: VARIABLE_RANGE(NUMBER_VARIABLES,2)
	END FUNCTION BINVAL
END INTERFACE


VAR_LOOP2: DO I=1,NUMBER_VARIABLES,1
	BIN_LOOP1: DO II=1,NUMBER_BIN,1
		MODELVALUE(II,I) = BINVAL(I,II,NUMBER_VARIABLES,NUMBER_BIN,VARIABLE_RANGE)
	END DO BIN_LOOP1
END DO VAR_LOOP2

END SUBROUTINE MODEL_VALUE
```

Appendix B.4: SUBROUTINE OPTIMUM_MODEL

```
!OPTIMUM_MODEL-----------------------------------------------------------
! THIS SUBROUTINE OPEN UP THE PPD FILE AND CALCULATES A POPLUATION OF FEASIBLE
! VARIABLE VALUES BASED ON THE PROBABILITY DISTRIBUTION OF THE PPD.
!
! VARIABLES:
!		I, II				DO LOOP COUNTERS
!		STATUS				I/O STATUS INDICATOR
!		SEED				RANDOM NUMBER GENORATOR SEED
!		N				RANDOMLY SELECTED BIN
!		GENERATIONS			NUMBER OF GENERATIONS
!		NUMBER_VARIABLES		NUMBER OF VARIABLES
!		NUMBER_BIN			NUMBER OF BINS IN PPD
!		POPULATION_SIZE				SIZE OF POPULATION
!		MODELVALUE			MODEL VALUE PER BIN
!		PPD				POSTERIOR PROBABILITY DENSITY
!		SELECTED			BOOLIAN VARIABLE FOR SELECTION
!
! INPUT:
!		GENERATIONS
!		NUMBER_VARIABLES
!		NUMBER_BIN
!		POPULATION_SIZE
!		MODELVALUE
!		FILE='PPD'
!
! OUTPUT:
!		FILE='OPTIMUM_MODEL'
!
! SUBROUTINES:
!		RANDOM_INITIALIZER
!
! FUNCTIONS:
!		URAND
!-------------------------------------------------------------------------

SUBROUTINE OPTIMUM_MODEL(GENERATIONS,NUMBER_VARIABLES,NUMBER_BIN,POPULATION_SIZE,MODELVALUE)

IMPLICIT NONE

INTEGER :: I, II, STATUS, SEED, N
INTEGER, INTENT(IN) :: GENERATIONS,NUMBER_VARIABLES,NUMBER_BIN,POPULATION_SIZE

REAL, INTENT(IN) :: MODELVALUE(NUMBER_BIN,NUMBER_VARIABLES)
REAL, ALLOCATABLE :: PPD(:,:),OP_MOD(:,:)

LOGICAL :: SELECTED

INTERFACE
	FUNCTION URAND(SEED)
		IMPLICIT NONE
		INTEGER :: SEED
		REAL :: URAND
		REAL, PARAMETER :: NORMALIZER=1.5258E-5
	END FUNCTION URAND

	SUBROUTINE RANDOM_INITIALIZER(SEED)
		IMPLICIT NONE
		INTEGER, INTENT(OUT) :: SEED
		INTEGER :: I
	END SUBROUTINE RANDOM_INITIALIZER
END INTERFACE

ALLOCATE(PPD(NUMBER_VARIABLES,NUMBER_BIN),STAT=STATUS)
	IF (STATUS /= 0) STOP "NOT ENOUGH MEMORY FOR PPD"
```

```
ALLOCATE(OP_MOD(POPULATION_SIZE,NUMBER_VARIABLES),STAT=STATUS)
        IF (STATUS /= 0) STOP "NOT ENOUGH MEMORY FOR OP_MOD"

! INITIALIZE RANDOM NUMBER GENERATOR
CALL RANDOM_INITIALIZER(SEED)

! OPEN THE PPD FILE AND READ IN THE PPD FOR THE FINAL GENERATION
OPEN (UNIT=10,FILE='PPD',STATUS='OLD',ACTION='READ',POSITION='REWIND',IOSTAT=STATUS)
        IF (STATUS>0) STOP "CAN NOT OPEN FILE: PPD"

!POSITION CURSOR AT LAST GENERATION IN PPD FILE
GEN_LOOP1: DO I = 0,GENERATIONS-1,1
        VAR_LOOP1: DO II = 1,NUMBER_VARIABLES,1
                READ(UNIT=10,FMT=*,ERR=40,IOSTAT=STATUS)
        END DO VAR_LOOP1
END DO GEN_LOOP1

VAR_LOOP2: DO I = 1,NUMBER_VARIABLES,1
        READ(UNIT=10,FMT=*,ERR=40,IOSTAT=STATUS) (PPD(I,II),II=1,NUMBER_BIN,1)
END DO VAR_LOOP2

CLOSE(UNIT=10)

! NORMALIZE PPD
VAR_LOOP4: DO I = 1,NUMBER_VARIABLES,1
        PPD(I,1:NUMBER_BIN:1) = PPD(I,1:NUMBER_BIN:1)/(SUM(PPD(I,1:NUMBER_BIN:1)))
END DO VAR_LOOP4

! PROCEED TO SELECT MODEL VARIABLES BASED ON THE FINAL PPD
POP_LOOP1: DO I = 1,POPULATION_SIZE,1
        VAR_LOOP3: DO II = 1,NUMBER_VARIABLES,1
                SELECTED = .FALSE.
                DO
                        N = ANINT(URAND(SEED)*REAL(NUMBER_BIN))
                                IF (N==0) N = 1
                                IF (N==(NUMBER_BIN+1)) N = NUMBER_BIN
                                IF (N>NUMBER_BIN) PRINT*,'ERROR IN OPTIMUM MODEL'
                        IF (URAND(SEED)<=PPD(II,N)) SELECTED = .TRUE.
                        IF (SELECTED) EXIT
                END DO
                OP_MOD(I,II) = MODELVALUE(N,II)
        END DO VAR_LOOP3
END DO POP_LOOP1

! WRITE OUTPUT OF PPD
OPEN (UNIT=10,FILE='OPTIMUM_MODEL',STATUS='REPLACE',ACTION='WRITE',POSITION='REWIND',IOSTAT=STATUS)
        IF (STATUS>0) STOP "CAN NOT OPEN FILE: OPTIMUM_MODEL"
POP_LOOP2: DO I = 1,POPULATION_SIZE,1
        WRITE(UNIT=10,FMT=*,ERR=40,IOSTAT=STATUS) (OP_MOD(I,II),II = 1,NUMBER_VARIABLES,1)
END DO POP_LOOP2
CLOSE(UNIT=10)

40 IF (STATUS > 0) PRINT*,'INPUT DATA ERROR'

DEALLOCATE(OP_MOD,STAT=STATUS)
        IF (STATUS /= 0) STOP "DEALLOCATE ERROR FOR OP_MOD"
DEALLOCATE(PPD,STAT=STATUS)
        IF (STATUS /= 0) STOP "DEALLOCATE ERROR FOR PPD"

END SUBROUTINE OPTIMUM_MODEL
```

Appendix B.5: SUBROUTINE NORMAL

```
!NORMAL-------------------------------------------------------------
! THIS SUBROUTINE DETERMINES THE FUZZY FITNESS OF EACH MEMBER OF A GENERATION
```

```
! AND PLACES THE FITNESS IN A AN ARRAY PM THAT IS POPULATION_SIZE LONG.
!
! VARIABLES:
!           I,II                        DO LOOP COUNTERS
!           STATUS                      ALLOCATION AND/OR IO STATUS NUMBER
!           POPULATION_SIZE                     POPULATIONS SIZE
!           NUMBER_OBJECTIVES           NUMBER OF OBJECTIVES
!           EXPERIMENTAL_DATA           EXPERIMENTAL DATA ARRAY
!           EXPERIMENTAL_ERROR          EXPERIMENTAL ERROR ARRAY
!           CALC_DATA                   UNSCALED CALCUALTED VALUES
!           PM                          BAYES SCALED FITNESS VECTOR
!           TOP_SCALE                   FUZZY FITNESS UPPER SCALE LIMIT (CONSTANT)
!           BOTTOM_SCALE                FUZZY FITNESS LOWER SCALE LIMIT (CONSTANT)
!           RMAX(:)                     LARGEST UNSCALED OBJ VALUE VECTOR
!           RMIN(:)                     SMALLEST UNSCALED OBJ VALUE VECTOR
!           TEMP1                       LARGEST TEMP UNSCALED OBJ VALUE
!           TEMP2                       SMALLEST TEMP UNSCALED OBJ VALUE
!           X1,Y1                       FUZZY POINT 1
!           X2,Y2                       FUZZY POINT 2
!           X3,Y3                       FUZZY POINT 3
!           X4,Y4                       FUZZY POINT 4
!           X5,Y5                       FUZZY POINT 5
!           X                           CALC_DATA VALUE FOR WHICH A FUZZY FITNESS IS NEEDED
!           AVERAGE                             AVERAGE FUZZY FITNESS FOR A POPULATION MEMBER
!           SLOPE                       SCALING SLOPE FOR BAYES SCALE
!           B                           Y INTERCEPT FOR BAYES SCALING
!           MAXIMUM                             MAXIMUM VALUE USED IN PPD SCALING
!           MINIMUM                     MINIMUM VALUE USED IN PPD SCALING
!           DIFFERENCE                  DIFFERENCE BETWEEN MAXIMUM AND MINIMUM
! INPUT:
!
           POPULATION_SIZE,NUMBER_OBJECTIVES,EXPERIMENTAL_DATA,EXPERIMENTAL_ERROR,CALC_DATA,SUMT
OTAL
! OUTPUT:
!           PM
! SUBROUTINES:
!           NONE
! FUNCTIONS:
!           AVG
!-----------------------------------------------------------------------

SUBROUTINE NORMAL(POPULATION_SIZE,NUMBER_OBJECTIVES,EXPERIMENTAL_DATA,&
EXPERIMENTAL_ERROR,CALC_DATA,PM)

IMPLICIT NONE

INTEGER :: I,II,STATUS
INTEGER, INTENT(IN) :: POPULATION_SIZE
INTEGER, INTENT(IN) :: NUMBER_OBJECTIVES

REAL, INTENT(IN) :: EXPERIMENTAL_DATA(NUMBER_OBJECTIVES),EXPERIMENTAL_ERROR(NUMBER_OBJECTIVES)
REAL, INTENT(IN) :: CALC_DATA(POPULATION_SIZE,NUMBER_OBJECTIVES)
REAL, INTENT(OUT) :: PM(POPULATION_SIZE)
REAL, PARAMETER :: TOP_SCALE = 1.0, BOTTOM_SCALE=0.0
REAL, ALLOCATABLE :: RMAX(:),RMIN(:)
REAL :: X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5,X,AVERAGE,SLOPE,B,MAXIMUM,MINIMUM,DIFFERENCE,TEMP1,TEMP2

INTERFACE
           FUNCTION AVG(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5,X)
                      REAL :: AVG
                      REAL, INTENT(IN) :: X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5,X
           END FUNCTION AVG
END INTERFACE

ALLOCATE(RMAX(NUMBER_OBJECTIVES),STAT=STATUS)
           IF (STATUS /= 0) STOP "NOT ENOUGH MEMORY FOR RMAX"
```

```
ALLOCATE(RMIN(NUMBER_OBJECTIVES),STAT=STATUS)
        IF (STATUS /= 0) STOP "NOT ENOUGH MEMORY FOR RMIN"

!ARRAY INITIALIZATION
PM = 0.0

! DETERMINING THE MOST UNFIT MEMBERS OF THE POPULATION FOR THE UPPER AND LOWER
! BOUNDS OF THE FUZZY LOGIC RULE SET
RMAX = EXPERIMENTAL_DATA+EXPERIMENTAL_ERROR
RMIN = EXPERIMENTAL_DATA-EXPERIMENTAL_ERROR
OBJ_LOOP2: DO I = 1,NUMBER_OBJECTIVES,1
        TEMP1 = MAXVAL(CALC_DATA(1:POPULATION_SIZE:1,I:I:1))
        TEMP2 = MINVAL(CALC_DATA(1:POPULATION_SIZE:1,I:I:1))
        IF (TEMP1>RMAX(I)) RMAX(I)=TEMP1
        IF (TEMP2<RMIN(I)) RMIN(I)=TEMP2
END DO OBJ_LOOP2

! DETERMINE THE FITNESS OF EACH MEMBER OF THE POPULATION USING THE FUZZY LOGIC
! RULE SET.  THE TOTAL FITNESS IS THE AVERAGE OF ALL THE FITNESS OF EACH OBJECTIVE
! FOR EACH MEMBER OF THE POPULATION.
POP_LOOP1: DO I = 1,POPULATION_SIZE,1
        AVERAGE = 0.0
        OBJ_LOOP3: DO II = 1,NUMBER_OBJECTIVES,1
                X1 = RMIN(II)
                Y1 = BOTTOM_SCALE
                X2 = EXPERIMENTAL_DATA(II)-EXPERIMENTAL_ERROR(II)
                Y2 = TOP_SCALE
                X3 = EXPERIMENTAL_DATA(II)
                Y3 = TOP_SCALE
                X4 = EXPERIMENTAL_DATA(II)+EXPERIMENTAL_ERROR(II)
                Y4 = TOP_SCALE
                X5 = RMAX(II)
                Y5 = BOTTOM_SCALE
                X = CALC_DATA(I,II)
                AVERAGE = AVERAGE+AVG(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5,X)
        END DO OBJ_LOOP3
        PM(I) = AVERAGE/REAL(NUMBER_OBJECTIVES)
END DO POP_LOOP1

DEALLOCATE(RMAX,STAT=STATUS)
        IF (STATUS /= 0) STOP "DEALLOCATE ERROR FOR RMAX"
DEALLOCATE(RMIN,STAT=STATUS)
        IF (STATUS /= 0) STOP "DEALLOCATE ERROR FOR RMIN"

END SUBROUTINE NORMAL
```

Appendix B.6: FUNCTION AVG

```
!AVG-------------------------------------------------------------
!This function is the heart of the fuzzy logic selection method.
!The points listed BELOW define the five major points in a
!fuzzy logic rule set:
!  1\          /5
!    \        /
!     \      /
!      2__3__4
! AVG determines between which two points the value of X lies
! and then calls POINT to determine the cooresponding value of Y
!
! VARIABLES:
!           AVG     VALUE OF Y SENT BACK CORRESPONDING TO X
!           X1,Y1   POINT 1
!           X2,Y2   POINT 2
!           X3,Y3   POINT 3
!           X4,Y4   POINT 4
```

```
!           X5,Y5     POINT 5
!           X         CALCULATED VALUE WHOSE FITNESS NEEDS TO BE DETERMINED
! INPUT:
!           X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5,X
! OUTPUT:
!           AVG
! SUBROUTINES:
!           NONE
! FUNCTIONS:
!           POINT
!-------------------------------------------------------------------------
FUNCTION AVG(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5,X)

IMPLICIT NONE

REAL :: AVG
REAL, INTENT(IN) :: X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5,X

INTERFACE
        FUNCTION POINT(X1,Y1,X2,Y2,X)
                REAL :: POINT,SLOPE,B
                REAL, INTENT(IN) :: X1,Y1,X2,Y2,X
        END FUNCTION POINT
END INTERFACE

IF (X1>X2.OR.X2>X3.OR.X3>X4.OR.X4>X5) THEN          !TESTING TO MAKE SURE THE FIVE POINTS ARE IN THE
PROPER ORDER
        PRINT*,'ERROR #1 IN AVG, POINTS OUT OF ORDER'
        READ(*,*)
        STOP
END IF

IF ((X>=X1).AND.(X<=X2)) THEN
        AVG = POINT(X1,Y1,X2,Y2,X)
ELSE IF ((X>=X2).AND.(X<=X3)) THEN
        AVG = POINT(X2,Y2,X3,Y3,X)
ELSE IF ((X>=X3).AND.(X<=X4)) THEN
        AVG = POINT(X3,Y3,X4,Y4,X)
ELSE IF ((X>=X4).AND.(X<=X5)) THEN
        AVG = POINT(X4,Y4,X5,Y5,X)
ELSE
        PRINT*,'ERROR #2 IN AVG, X OUT OF RANGE'
        READ(*,*)
        STOP
ENDIF

END FUNCTION AVG
```

Appendix B.7: FUNCTION BINVAL

```
!BINVAL-------------------------------------------------------------
! GIVEN A BIN NUMBER, THIS FUNCTION RETURNS THE AVERAGE VALUE
! OF THE BIN IN TERMS OF ITS REAL VARIABLE VALUE BETWEEN THE
! RANGES SPECIFIED IN VARIABLE_RANGES.
! VARIABLES:
!           VARID                   ID OF VARIABLE TYPE
!           BINID                   ID OF BIN POSITION
!           NUMBER_VARIABLES        NUMBER OF VARIABLES
!           NUMBER_BIN              NUMBER OF BINS
!           BINVAL                  REAL VALUE OF CENTER OF BIN
!           MAXIMUMVAR              MAXIMUM VALUE IN VARIABLE RANGE
!           MINIMUMVAR              MINIMUM VALUE IN VARIABLE RANGE
!           SLOPE                   SLOPE OF LINE INTERCEPT CONVERSION
!           B                       Y INTERCEPT OF LINE CONVERSION
!           Y1                      UPPER VALUE OF BIN
```

```
!             Y2                              LOWER VALUE OF BIN
!             VARIABLE_RANGE                  ARRAY CONTAINING VARIABLE RANGES.
! INPUT:
!             VARID,BINID,NUMBER_VARIABLES,NUMBER_BIN,VARIABLE_RANGE
! OUTPUT:
!             BINVAL
! SUBROUTINES:
!             NONE
! FUNCTIONS:
!             NONE
!-------------------------------------------------------------------------
FUNCTION BINVAL(VARID,BINID,NUMBER_VARIABLES,NUMBER_BIN,VARIABLE_RANGE)

IMPLICIT NONE

INTEGER,INTENT(IN) :: VARID,BINID,NUMBER_VARIABLES,NUMBER_BIN

REAL :: BINVAL,MAXIMUMVAR,MINIMUMVAR,SLOPE,B,Y1,Y2
REAL, INTENT(IN) :: VARIABLE_RANGE(NUMBER_VARIABLES,2)

MAXIMUMVAR = VARIABLE_RANGE(VARID,2)
MINIMUMVAR = VARIABLE_RANGE(VARID,1)

SLOPE = (MAXIMUMVAR-MINIMUMVAR)/(REAL(NUMBER_BIN))
B = MINIMUMVAR

Y1 = SLOPE*REAL(BINID)   + B
Y2 = SLOPE*REAL(BINID-1) + B

BINVAL = (Y1+Y2)/2.0

END FUNCTION BINVAL
```

Appendix B.8: SUBROUTINE RANDOM_INITIALIZER

```
!RANDOM_INITIALIZER-------------------------------------------------------------
! THIS SUBROUTINE RETURNS A SEED GENERATED BY ACCESSING THE SYSTEM CLOCK AND THEN
! CALLS THE RANDOM_SEED() BUILT-IN SUBROUTINE A NUMBER OF TIMES THAT IS A FUNCTION OF THE
! SEED VALUE TO ENSURE THAT THE RANDOM NUMBER GENERATOR (URAND) REMAINS RANDOM.
! THIS SUSROUTINE SHOULD ONLY BE CALLED ONCE IN THE RUNNING OF A PROGRAM.
!
! VARIABLES:
!             I                       DO LOOP COUNTER
!             SEED                    RANDOM NUMBER SEED
! INPUT:
!             SEED
! OUTPUT:
!             SEED
! SUBROUTINES:
!             NONE
! FUNCTIONS:
!             NONE
!-------------------------------------------------------------------------

SUBROUTINE RANDOM_INITIALIZER(SEED)

IMPLICIT NONE

INTEGER, INTENT(OUT) :: SEED
INTEGER :: I

CALL SYSTEM_CLOCK(SEED)
SEED = INT(REAL(SEED)/1.0E6)
SEED_LOOP: DO I = 1,SEED,1
             CALL RANDOM_SEED()
```

```
END DO SEED_LOOP

END SUBROUTINE RANDOM_INITIALIZER
```

Appendix B.9: FUNCTION URAND

```
!URAND-------------------------------------------------------------
! THIS FUNCTION RETURNS A RANDOMLY GENERATED NUMBER BETWEEN 0 AND 1
! AS THE VARIABLE URAND
!
! THE CALLING PROGRAM SHOULD CONTAIN THE FOLLOWING LINES IN ORDER TO
! PROPERLY INITIALIZE THE RANDOM NUMBER GENERATOR.  SEED IS AN INTEGER:
!
!             CALL SYSTEM_CLOCK(SEED)
!             SEED = INT(REAL(SEED)/1.0E6)
!             SEED_LOOP: DO I = 1,SEED,1
!                         CALL RANDOM_SEED()
!             END DO SEED_LOOP
!
! NOTE: IF A REAL NUMBER BETWEEN 0 AND 1 IS NOT RETURNED, SEE NOTES IN CODE BELOW
!
! VARIABLES:
!             URAND               REAL NUMBER BETWEEN 0 AND 1
!             SEED                RANDOM NUMBER SEED
!             NORMALIZER          VARIABLE TO ENSURE THAT URAND FALLS BETWEEN 0 AND 1
!                                 THIS MAY BE SYSTEM DEPENDENT
! INPUT:
!             BE SURE TO CALL RANDOM_SEED FIRST
! OUTPUT:
!             URAND   RANDOM NUMBER
! SUBROUTINES:
!             NONE
! FUNCTIONS:
!             NONE
!-------------------------------------------------------------------------

FUNCTION URAND(SEED)

IMPLICIT NONE

INTEGER :: SEED

REAL :: URAND
REAL, PARAMETER :: NORMALIZER=1.5258E-5

CALL RANDOM_NUMBER(URAND)

! WHEN RUNNING THIS CODE ON MY POWER MACINTOSH G3, THE INSTRINSIC FUNCTION
! RANDOM_NUMBER WAS PROVIDING RANDOM NUMBERS BETWEEN 0 AND 1.5258E-5
! IF YOU RUN THIS CODE ON YOUR COMPUTER AND THE URAND IS NOT RANDOM THEN PERHAPS
! THE DEFAULT RANDOM RANGE ON YOUR COMPUTER IS DIFFERENT AND YOUR NORMALIZER NEEDS
! TO BE ADJUSTED

URAND = URAND/NORMALIZER

! THIS SAFETY STEP IS PROVIDED SO THAT IN THE EVENT THAT THE NORMALIZER IS NOT
! CORRECT, THE WORST THAT CAN HAPPEN IS THAT URAND IS NOT RANDOM.

IF (URAND>1.0) URAND = 1.0
IF (URAND<0.0) URAND = 0.0

END FUNCTION URAND
```

Appendix B.10: FUNCTION POINT

```
!POINT-------------------------------------------------------------
!GIVEN TWO POINTS (X1,Y1) AND (X2,Y2) AND GIVEN A VALUE OF X
!BETWEEN X1 AND X2 INCLUSIVE, THE REAL FUNCTION POINT SENDS BACK
!THE CORRESPONDING VALUE OF Y.
!
! VARIABLES:
!           X1,Y1     POINT 1
!           X2,Y2     POINT 2
!           X         X VALUE FOR WHICH Y WILL BE FOUND
!           POINT     VALUE OF Y FOR X
!           SLOPE     SLOPE BETWEEN POINTS (X1,Y1) AND (X2,Y2)
!           B         Y INTERCEPT OF LINE WITH SLOPE AND INTERSECTING (X1,Y1)
! INPUT:
!           X1,Y1,X2,Y2,X
! OUTPUT:
!           POINT
! SUBROUTINES:
!           NONE
! FUNCTIONS:
!           NONE
!-------------------------------------------------------------------------
FUNCTION POINT(X1,Y1,X2,Y2,X)

IMPLICIT NONE

REAL :: POINT,SLOPE,B
REAL, INTENT(IN) :: X1,Y1,X2,Y2,X

IF (X1==X2) THEN              ! TO AVOID A DIVISION BY ZERO
          POINT = Y1
ELSE
          SLOPE = (Y2-Y1)/(X2-X1)
          B  = Y1-SLOPE*X1
          POINT = SLOPE*X+B
ENDIF

END FUNCTION POINT
```

Appendix B.11: FUNCTION BINNER

```
!BINNER-------------------------------------------------------------
!THE PURPOSE OF THE INTEGER FUNCTION BINNER IS TO TAKE A REAL VALUE AND
!DETERMINE ITS BIN NUMBER WITHIN A SET RANGE OF VALUES.
!
! VARIABLES:
!           BINNER                  BIN NUMBER FOR THE VARIABLE
!           VARID                   THE VARIABLE ID NUMBER TO BE BINNED
!           NUMBER_BIN              NUMBER OF BINS
!           NUMBER_VARIABLES        NUMBER OF VARIABLES
!           VAR                     VARIABLE VALUE TO BE BINNED
!           VARIABLE_RANGE                  UPPER AND LOWER BOUNDS OF BINNING
!           RANGE                   SIZE OF BINNING RANGE
!           SLOPE                   # BINS / RANGE
!           B                       Y INTERCEPT OF LINE FOR BINS AND RANGE
! INPUT:
!           VARID,VAR,NUMBER_BIN,NUMBER_VARIABLES,VARIABLE_RANGE
! OUTPUT:
!           BINNER
! SUBROUTINES:
!           NONE
! FUNCTIONS:
!           NONE
!-------------------------------------------------------------------------
```

```
FUNCTION BINNER(VARID,VAR,NUMBER_BIN,NUMBER_VARIABLES,VARIABLE_RANGE)

IMPLICIT NONE

INTEGER :: BINNER
INTEGER, INTENT(IN) :: VARID,NUMBER_BIN,NUMBER_VARIABLES
REAL, INTENT(IN) :: VAR,VARIABLE_RANGE(NUMBER_VARIABLES,2)
REAL :: RANGE,SLOPE,B

! EQUATION OF A LINE WHERE THE MINIMUM VARIABLE VALUE AND THE
! BIN # 0 ARE ONE POINT AND THE MAXIMUM VARIABLE VALUE AND THE
! TOTAL NUMBER OF BINS ARE THE OTHER POINT

RANGE = VARIABLE_RANGE(VARID,2)-VARIABLE_RANGE(VARID,1)

!SLOPE = NUMBER_BIN - 0 / MAXVAR - MINVAR
SLOPE = REAL(NUMBER_BIN)/RANGE

! Y INTERCEPT
B     = -SLOPE*VARIABLE_RANGE(VARID,1)

BINNER = INT(SLOPE*VAR+B)+1


! ERROR CORRECTION IN CASE OF A MISTAKE
IF (BINNER==(NUMBER_BIN+1)) BINNER=NUMBER_BIN

IF (BINNER>NUMBER_BIN.OR.BINNER<1) THEN
        PRINT*,'ERROR IN BINNER',BINNER
        READ(*,*)
        STOP
END IF

END FUNCTION BINNER
```